Parallel Shuffling and its Application to Prêt à Voter

Kim Ramchen
kramchen@gmail.com

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

Abstract

We consider the problem of *verifiable parallel shuffling* in which the same shuffle is simultaneously performed on two or more lists of input ciphertexts, each list encrypted under a different key. We present three parallelisations of shuffle proofs from different paradigms. The properties of each protocol are analyzed and contrasted, and their suitability for electronic voting discussed. We show how parallel shuffling solves the problem of verifiable distributed ballot construction in the Prêt à Voter electronic voting scheme. In conjunction with the use of a new cryptographic primitive for partially-homomorphic addition, the incorporation of parallel shuffling is shown to produce schemes with superior privacy properties to existing protocols. We additionally present several original attacks on Prêt à Voter and demonstrate that our modified schemes are immune.

1 Introduction

A shuffle is a permutation and re-encryption of a set of input ciphertexts. A mix-net is a series of chained servers each of which applies a shuffle to some input ciphertexts, before passing the output to the next server. In an electronic voting context, one possible use of a mix-net is to untraceably shuffle encrypted votes prior to decryption, ensuring an individual's vote is private.

In recent years much effort has gone into designing efficient verifiable shuffle arguments, for use in mix-nets. There have been a number of approaches for verifiably shuffling a list of homomorphic encryptions, beginning with the classic proof by Sako and Kilian [SK95]. Later Furukawa and Sako [FS01] used characterisations of permutation matrices to achieve a much more efficient proof, a technique extended by Groth [GL07]. A third paradigm by Neff [Nef01], for El Gamal ciphertexts, is based upon the observation that a polynomial is invariant under a permutation of its roots, and makes use of the fact that the number of roots of a polynomial over $\mathbb{Z}_q$ (where q is prime) is bounded by the degree. This technique has been refined in [Nef04] and later extended to arbitrary homomorphic cryptosystems in [Gro05].

In this paper, we focus on constructing efficient protocols for *verifiable parallel shuffling*, in which a mix server verifiably applies the same shuffle to several input vectors. Although the problem has been examined in [Gro05] and [Nef04] in the case that all ciphertexts are encrypted under the same public key, our work is the first to explicitly address the case that the input vectors are each encrypted under distinct public keys. We present one solution for verifiable parallel shuffling, which is a generalisation of the classic shuffle proof of Sako and Killian. Although the generalisation is straightforward, we are not aware of its being published before. We sketch another solution based on [GL07]. Appendix B contains a final scheme based on a proof by Neff [Nef04], suitable for El Gamal ciphertexts. The latter schemes are considerably more efficient but also more complicated. The parallelisations maintain the same privacy properties as the original protocols. For example, the parallelisation of Neff's proof is shown to be permutation hiding as defined in [NSNK06].

1.1 Applications: Prêt à Voter

Prêt à Voter is an end to end electronic voting system, which satisfies the requirements of correctness, privacy and voter-friendliness [RS06, Rya08]. We have chosen to focus on Prêt à Voter because its user interface is easily adapted to expressive voting schemes such as Single Transferable Vote (STV), Range Voting and Borda. These are the main applications of the ballot construc-

tions described in this work. In Prêt à Voter, as in other verifiable voting systems, achieving coercion-resistance even in single-value elections such as First Past the Post is a challenging problem. Although the basic design of Prêt à Voter is receipt-free (meaning that a voter cannot prove how they voted), various subtle coercion attacks have been demonstrated against particular versions [XSHT08, RT09]. When allowing ranked-voting systems such as STV, the problem becomes much harder again [Hea07, XSHT08, BMN+].

In this paper we illustrate a number of new weaknesses relating to ballot construction in existing schemes that could result in coercible or incomplete elections. Coercible means that voters can prove how they voted and hence sell their votes; incomplete means that the tally may not be exactly correct even when the proof transcript is valid. We then present efficient alternatives that avoid such problems. In particular we present a primitive based on El Gamal that allows homomorphic modular addition of a plaintext message with an encrypted message at no cost. This construction is perfectly suited for tabulating votes in single-value elections, eliminating the possibilities of incompleteness or coercion that exist in current schemes.

We show how parallel shuffling dovetails with this approach, and more generally how it leads to verifiable ballot construction in both single-value and ranked-voting elections, which in turn yields improved coercion-resistance.

Structure of the Paper We devote Section 2 to the cryptographic background that the rest of the paper utilises, including the modular addition primitive that is employed in the discussion of Prêt à Voter. In Section 3 we present a parallelisation of a verifiable shuffle protocol, with a sketch of two more and a comparison of their properties and suitability for Electronic Voting. In Section 4 we discuss weaknesses with Prêt à Voter schemes and show how parallel shuffling resolves them.

2 Cryptographic Preliminaries

2.1 Homomorphic Encryption

We require a cryptosystem with homomorphic encryption, that is the encryption function *homomorphically* maps from a message (plaintext) space to a ciphertext space. This simply means that some public operation can be performed on ciphertexts which reflects an operation on the private underlying messages. We will discuss protocols which either specifically require additive homomorphism (typically in a voting context), or in which the homomorphism does not matter (e.g. in shuffling).

2.1.1 Example - Additive El Gamal

Let q, p be large primes with $q|p-1$. Let $\mathcal{G}$ be a subgroup of $\mathbb{Z}_p^*$ of order q with generator g. The private key is s chosen from $[0, q-1]$. Let $h = g^s$, then the public key is $\langle p, q, g, h\rangle$. Encryption and decryption of a message $m \in [0, q-1]$ with randomness r uniformly selected from $[0, q-1]$ are given by:

$$E_{pk}(m; r) = (g^r, g^m h^r)$$
$$D_{sk}((x, y)) = \log_g \frac{y}{x^s} = m$$

We get an additive homomorphism:

$$E_{pk}(m_0; r_0) \cdot E_{pk}(m_1; r_1) = (g^{r_0+r_1},\ g^{m_0+m_1} h^{r_0+r_1}) = E_{pk}(m_0 + m_1; r_0 + r_1)$$

We also get homomorphic scalar multiplication:

$$(g^r, g^m h^r)^c = (g^{rc}, g^{mc} h^{rc})$$

Note in this scheme decryption is only practical for small m owing to the difficulty of the discrete log problem, however this is an acceptable restriction in many applications, including electronic voting.

The Paillier [Pai99] cryptosystem also has an additive homomorphism, without requiring a small message space.

2.2 Scalar Homomorphism Modulo N

In this section we present a scheme in which an unencrypted value can be added to an encrypted one, producing a ciphertext which is the sum of the original two values modulo a public integer N.

The setup is the same as for El Gamal encryption over $\mathbb{Z}_p^*$, with g an element of (large prime) order q. Suppose $N \mid q-1$. Then an element $\alpha \in \mathbb{Z}_q^*$ of order N in $\mathbb{Z}_q^*$ exists and is easy to construct.[1] In [HSvV09] this root of unity was used to construct an efficient proof of

[1] One way to do so is to find a generator r of $\mathbb{Z}_q^*$ (*i.e.* the exponent group) and set $\alpha = r^{\frac{q-1}{N}}$. Finding r is not difficult providing $q-1$ is easily factored and does not have too many small factors - since a random element in $\mathbb{Z}_q^*$ is a generator with probability $\frac{\phi(q-1)}{q-1}$.

rotation. However it also yields a "partial" homomorphism, which allows us to compose a plaintext message with another encrypted message $(\mathrm{mod}\ N)$. We have, taking superscripts of α modulo N:

$$(g^r, g^{\alpha^l} h^r)^{\alpha^m} = (g^{r'}, g^{\alpha^{l+m}} h^{r'}), \text{ that is:}$$
$$\alpha^m \otimes E(\alpha^l) = E(\alpha^{l+m}) :$$

Of course to allow decryption we require relatively small N due to the necessity of extracting a message m from g^{α^m}.

When we actually use this, we will want N to be the number of candidates in the election. This should still be a reasonable constraint to satisfy.

3 Parallel Shuffling

In this section we formalise verifiable parallel shuffling and give one detailed example, based on a classic shuffle proof. Although the extension is straightforard, we have not seen it elsewhere in the literature. We include it here for the sake of having a concrete, easily understood example. Two other, more efficient but more complicated solutions, are contained in Section 3.3 and Appendix B. We conclude this section with a comparison of their properties, and discuss their suitability for electronic voting generally, though our specific applications are for Prêt à Voter.

3.1 Definition of parallel shuffling

Informally, we wish to be able to prove that the same shuffle π has been applied to each vector of input ciphertexts $\{e_i^j\}$. We refer to this problem as $Shuf_{Par}$, and define it formally as follows:

Definition $Shuf_{Par}$

Let $E^j : 1 \leq j \leq J$ be semantically secure homomorphic encryption schemes, each with a corresponding randomizer space $\mathcal{R}^j$.

Common input: J vectors of input ciphertexts:

$$\{e_i^1\}, \ldots, \{e_i^J\} \text{ with } 1 \leq i \leq k$$

and J vectors of corresponding output ciphertexts:

$$\{E_i^1\}, \ldots, \{E_i^J\} \text{ with } 1 \leq i \leq k$$

Prover's input: A permutation $\pi \in S_k$, re-encryption factors $\beta_i^j \in \mathcal{R}^j, 1 \leq j \leq J$ so that for each j:

$$E_i^j = e_{\pi(i)}^j \cdot E^j(0; \beta_i^j) : 1 \leq i \leq k$$

To be proved: That π, defined above, exists.

We now give three different solutions to $Shuf_{Par}$, each based on a different shuffle proof.

3.2 Parallel Sako-Killian

The protocol is a parallelisation of a proof in [SK95] which is itself based on classic technique for zero knowledge proofs introduced in [GMW91]. The prover begins by generating a random shuffle σ and applying it to each list of input ciphertexts, producing intermediate shuffled lists. The verifier challenges the prover to reveal either the permutation that takes the input lists to the intermediate lists, or the permutation that takes the intermediate lists to the outputs. A more formal protocol description is given in Algorithm 1. The security properties and proofs are very similar to those in [SK95].

We use the notation iid to refer to random variables that are independent and identically distributed.

Algorithm 1 Parallel Sako-Killian shuffle proof

for Repeat t times **do**

$\mathcal{P}$ picks $\sigma \overset{d}{\sim} U(S_n)$, γ_i^j iid $U(\mathcal{R}^j)$ and sends:

$$I_i^j = e_{\sigma(i)}^j \cdot E^j(0; \gamma_i^j) \tag{1}$$

$\mathcal{V}$ challenges with bit b.

if $b = 0$ **then**

$\mathcal{P}$ reveals σ, γ_i^j

$\mathcal{V}$ accepts if (1) holds.

end if

if $b \neq 0$ **then**

$\mathcal{P}$ reveals $\phi = \sigma^{-1}\pi, \delta_i^j = \beta_i^j - \gamma_{\sigma^{-1}\pi(i)}^j$

$\mathcal{V}$ accepts if:

$$E_i^j = I_{\phi(i)}^j \cdot E^j(0; \delta_i^j)$$

end if

end for

Theorem 3.1 *Algorithm 1 is a sound, complete and zero knowledge protocol for* $Shuf_{Par}$.

Proof A straightforward extension of the proof in [SK95].

A disadvantage of this approach is that the communication complexity is $O(tnJ)$.

3.3 Permutation Matrices

The most efficient shuffle proof based on permutation matrices is [GL07], which, using commitment schemes is actually an argument.[2] One advantage of the commitment approach is that it is easily and efficiently parallelised. At a conceptual level, once a permutation π has been committed, any set of input-ouput ciphertexts that has been shuffled by π needs only to be linked to the commitments. The exact mechanics of how this is done would require reproduction of the arguments in [GL07], however it suffices to say that every additional shuffle to be proved requires just one extra message to be sent. The property of being parallelisable is discussed in [Gro05] for the shuffle argument there (which was actually based on the Neff-paradigm), however the parallelisation technique for [GL07] is identical, so the reader is referred to [Gro05].

3.4 Comparison of Parallel Shuffles

Appendix B contains a description of how to extend Neff's polynomial approach [Nef04] to parallel shuffling.

The choice of which parallel shuffle proof to use depends somewhat on the application. Ignoring efficiency, the parallelisation in Section 3.2 of Sako and Killian's classic proof is very attractive, as it is simple, generic, attains arbitrary soundness given enough rounds, and is perfect zero knowledge. However in practice efficiency is usually paramount and perfect zero knowledge is actually an unnecessary requirement. In fact, as the input to the shuffle proof is encrypted and the semantic security of encryption schemes themselves depend upon computational indistinguishability assumptions, computational zero knowledge is sufficient.

The permutation matrix approach sketched in Section 3.3, based on [GL07], is the most efficient, and is computational zero knowledge. The use of an integer commitment scheme there allows an argument based on a neat characterisation of a permutation matrix over $\mathbb{Z}$. The round complexity is only 3, desirable in the non-interactive case as it reduces the computational work in the Fiat-Shamir heuristic etc. A disadvantage is that it relies on *computationally binding* rather than *statistically binding* commitment schemes, so it is technically not a *proof*, but an *argument*. This means that the soundness and hence trustworthiness of the protocol is based on computational constraints.

In contrast, the parallelisation in Appendix B of Neff's proof [Nef04] is a true proof, as it is *unconditionally sound*. It is not computational zero knowledge, however it satisfies the weaker privacy notion of permutation hiding, which is sufficient for our purposes.

One important disadvantage of [GL07] for electronic voting is that using commitment schemes that are only computationally binding, results in an election protocol that is not *universally verifiable*. The integrity of the election relies upon a good source of randomness, either a common random string, a random oracle (implemented in practice with a strong hash function), or a value jointly generated by some authorities (in which case it must be assumed they don't all collude). Although Neff's protocol relies on similar assumptions, it is only for the privacy, not the integrity, of the election. The permissibility of universal verifiability is a clear advantage of *"Shuffle of ElGamal Pairs"* [Nef04] over shuffle protocols based on computationally binding commitments, and hence of the parallelisation in Appendix B over the parallelisation in Section 3.3.

4 Applications to Prêt à Voter

This section is divided into three parts. In the first we present an overview of the important constructions that underlie Prêt à Voter, though the reader is referred to [RS06, Rya08] for full details. In the second we discuss a number of weaknesses with existing Prêt à Voter schemes, and finally we show how parallel shuffling can efficiently solve them.

Notation Let v be the number of voters, and n the number of candidates. We will use $[\![x]\!]$ to denote an encryption of x, and the following notation for homomorphic encryption:

$$[\![(m_1)]\!] \oplus [\![(m_2)]\!] = [\![(m_1 + m_2)]\!]$$
$$[\![(m)]\!] \otimes c = [\![(mc)]\!]$$

4.1 Overview of Prêt à Voter

A cornerstone of the Prêt à Voter voter-verified system is the construction of the ballot form. A ballot form has a list of candidates printed down the left side, which is one

[2] The distinction and why it is important is discussed in Section 3.4.

Slytherin	
Gryffindor	
Hufflepuff	$\times$
Ravenclaw	
$[\![s]\!]_{Reg}$	$[\![s]\!]_{Tel}$

Figure 1: Single Value Election. Suppose the canonical ordering is alphabetical. The *cyclic shift* is 3, so $\Theta_{Tel} = [\![3]\!]_{Tel}$. The *checkmark index value* is $r = 2$. The voter's *preference index* with respect to the canonical order is hence $r + s = 2 + 3 = 1 \bmod 4$.

of n possible cyclic shifts of some public canonical candidate ordering (say an alphabetical list of candidates). At the bottom of the right side is a cryptographic "onion" encoding of the cyclic shift. In the booth, the voter marks a cross against their preferred candidate. Fig. 1 shows a completed ballot. Our conventions are that candidates are numbered from 0 to $n-1$, cyclic shifts push the candidate names upward, and voter checkmark positions are numbered downwards starting from 0.

The voter then detaches the right side from the left, discards the left side and retains only the right side of the form - known as a "receipt". Casting their vote consists of placing their receipt under a reader which records the index position r of the checkmark and the cryptographic "onion" Θ_{Tel} at the bottom of the right side.[3] The pair (r, Θ_{Tel}) is sent to a publicly visible bulletin board (BB), and the voter checks that it appears there. The voter's receipt is then digitally signed and they retain it.

The clever construction of the receipt makes the scheme:

1. *Voter-verified*. The voter can verify their vote is *cast as intended* simply by ensuring that the pair (r, Θ_{Tel}) that resides on their receipt corresponds to that displayed on the BB. The correctness of generation of ballot forms, ensures that the receipt is a true representation of the voter's preference.

2. *Receipt-free*. It is impossible for a *coercer* to learn how a voter voted, even if the voter is under the coercer's control. The plaintext on the voter's receipt is only the index value r, which could have been with respect to any possible cyclic shift of a canonical ordering. Therefore a a voter's receipt reveals nothing to a coercer.[4]

[3] In this paper an "onion" is a simple ciphertext.

[4] Providing the coercer doesn't obtain the discarded left strips, otherwise in theory a coercer could associate both sides of a ballot form from physical markings, and trivially learn the vote.

4.1.1 Anonymous Tabulation by Re-encryption

We have discussed why the scheme has the cast as intended property, in this section we discuss why the scheme has the *counted as cast* property.

Once the votes appear on the BB, they must be decrypted, so that the result of the election can be determined. However due to the public nature of postings to the BB, it is not satisfactory to simply decrypt the voters' receipts as they appear on the BB, or else a coercer could obtain a receipt from a voter and then note what the decrypted vote is. The solution is to pass the receipts through a *re-encryption mix-net*.[5] The mix-net consists of a number of parties, each of whom shuffles the input (here receipts) according to a private permutation, then proves the output really is the input permuted. Provided that at least one of the parties is honest, that is does not reveal their permutation, the privacy of the election will be preserved.

One catch is that the receipts are of the form $(r, [\![s]\!]_{Tel})$. We cannot simply pass the receipt as a pair through the mix, since the r-value is invariant (even though the ciphertext is re-encrypted and hence untraceable - the r-value may allow tracing of votes through the mix). The solution is to absorb the r-value into the onion via homomorphic encryption:

$$[\![r]\!]_{Tel} \oplus [\![s]\!]_{Tel} = [\![r+s]\!]_{Tel}$$
($[\![r]\!]_{Tel}$ is formed with zero randomness).

The single ciphertext is easily amenable to mixing, after which a threshold set of decryption tellers decrypt the output, recovering $r + s$. This value $\pmod n$ is precisely the index of the voter's preference in the public canonical ordering of candidates.

4.1.2 Generation of Encrypted Ballot Forms

We require that no single entity determines the encrypted candidate order that will appear on a voter's ballot form. This is because if a single authority generated $[\![s]\!]_{Tel}$, they would be able to decrypt a voter's receipt when it is posted (pre-mixing). The solution in [RS06] is to use a set of l clerks who will, for every ballot form, jointly generate the seed value, s, that will be used to construct the cyclic shift of candidates that will be printed on that form.[6] All clerks would have to collude to determine s

[5] In this paper when we refer to a "mix" or "shuffle", we will always mean a re-encryption mix-net.

[6] The registrars will calculate the cyclic shift as $s \pmod n$, and print the corresponding candidate order on the form.

from $[\![s]\!]_{Tel}$. Also s must be encrypted under the public key of the *registrars* - the authorities responsible for printing the candidate ordering on a ballot form. The joint generation of s is done as follows. For each tentative ballot in a batch, the jth clerk is passed the following pair by the $j-1$th clerk:

$$([\![s_{j-1}]\!]_{Reg},\ [\![s_{j-1}]\!]_{Tel})$$

She generates a random value $\bar{s}$ and outputs:

$$([\![s_j]\!]_{Reg},\ [\![s_j]\!]_{Tel}) =$$
$$([\![s_{j-1}]\!]_{Reg} \oplus [\![\bar{s}]\!]_{Reg},\ [\![s_{j-1}]\!]_{Tel} \oplus [\![\bar{s}]\!]_{Tel}).$$

These onion pairs are then mixed, and passed to the next clerk. The result after all l clerks are finished, is the pair:

$$([\![s]\!]_{Reg},\ [\![s]\!]_{Tel})$$

This registrars' onion and tellers' onion can now be printed to the bottom left and right corners respectively of a blank ballot form. The Teller's onion is covered with a scratch strip or similar, then the form is passed to the registrar for decryption. The registrar decrypts the registrar's onion (on the left side) and prints the candidates' names in order. The ballot is now available for auditing, to check that both onions and the plaintext candidate order match. If it is not audited, it can be voted on, after which the voter destroys the left side with the candidate list and the registrar's onion, then removes the scratch strip to expose the tellers' onion. The right side, with the tellers' onion and voter's checkmark, is scanned and posted on the bulletin board. Covering the tellers' onion prevents anyone, including the Registrar, from remembering its value and the corresponding candidate order, and finding them later on the bulletin board.

4.1.3 Accommodating Ranked Voting

The demands of accommodating single-value elections (e.g. FPTP) and ranked-voting elections (e.g. STV) in Prêt à Voter are different [Hea07, XSH+07]. The former requires the reconstruction of a vote from a receipt with a plaintext *index* r and encrypted candidate permutation σ, while the latter requires reconstruction from a plaintext *index permutation* π_r and encrypted candidate permutation σ. That is, the voter's actual *preference permutation* μ with respect to a canonical candidate ordering is $\pi_r\sigma$. Fig. 2 shows the ranked-voting case.

We note that obtaining coercion-resistance in ranked-voting elections when plaintext votes are revealed is inherently difficult due to the *Italian Attack* [BT94]. The attack exploits the fact that the number of all possible votes is enormously larger than the number of actual votes in an election. By assigning a particular preference permutation to a voter, a coercer can determine whether that voter obeyed simply by observing whether the permutation appears or not in the revealed votes. Moreover the fact that a coercer desires preferred candidate(s) to be elected scarcely hinders their coercion leverage, as discussed in [BMN+].

Ravenclaw	4
Hufflepuff	3
Gryffindor	1
Slytherin	2
$\langle s \rangle_{Reg}$	$\langle s \rangle_{Tel}$

Figure 2: Ranked Voting Election. Suppose the canonical ordering is alphabetical. The *candidate permutation* is $\sigma = \binom{\text{G H R S}}{3\,2\,1\,4}$. The *index permutation* is $\pi_r = \binom{1\,2\,3\,4}{4\,3\,1\,2}$. The voter's *preference permutation* is hence $\mu = \pi_r\sigma = \binom{\text{G H R S}}{1\,3\,4\,2}$.

4.2 Weaknesses in Ballot Construction

In this section we discuss a number of weaknesses in Prêt à Voter associated with ballot construction in existing schemes that result in either coercion, or elections that are incomplete. We then show how parallel shuffling is an efficient solution to these problems.

4.2.1 Verifiable Equivalence of Ballot Onions

To ensure the correctness of ballot forms, the equivalence of $[\![s]\!]_{Reg}$ and $[\![s]\!]_{Tel}$ must be verifiable/auditable.

In [Rya08] the joint generation of onions and auditing are done simultaneously, by having all onions published in a single step. Namely, to form w ballots, each of l clerks publish a column of w pairs, yielding a $w \times l$ matrix. In each row, auditors select a proportion (say half) of the pairs in each row, and require the clerks to decrypt them. Assuming no clerk is found to be cheating, the unaudited pairs in each row are homomorphically summed, yielding a single onion pair for each ballot.

Although highly convenient, this cut and choose protocol has the disadvantage that if only k clerks are honest (i.e. keep their plaintexts secret), then the other clerks in collusion learn the final seed values, and hence candidate permutations of a proportion of $\frac{1}{2^k}$ ballots. For very small k, this may lead to practical coercion, if a coercer colludes with the dishonest clerks.

Zhe (Joson) Xia [Xia09] suggested a simple solution to this problem: each clerk should produce a column with *two* different w values in each row, then one should be chosen at random for audit while the other is used to contribute to the onion. In Section 4.3 we present a different verifiable distributed ballot construction based on parallel shuffling. Both solutions require all clerks to collude to discover any s values. In our solution there is a negligible probability that a cheating ballot construction authority can construct a convincing proof; in Xia's there is a higher probability, though the probability is low enough that it would probably make no difference in a real election.

4.2.2 Unrestricted Seed Values

In this section we discuss why the approach of having the seed s generated from the entire plaintext space leads to aspects of incompleteness and coercibility. Recall that the plaintext space is very large, being the size of the prime-order subgroup that El Gamal runs in.

Incompleteness Observe that for the absorption of the index value r in the tabulation phase to work, we have to avoid overflow in the composition with the seed value s. Recall that n is the number of candidates, and let $0, \ldots, q-1$ be the (large) range of plaintexts. In the version given in Section 4.1.1, the value of the s that is jointly generated must not lie in $\{q-n, \ldots, q-1\}$. If it does, it is possible that $r + s > q$, which means that the decrypted candidate number will come out as $(r + s \bmod q) \bmod n$, which is not in general equal to the correct value, $r + s \bmod n$.

In general, regardless of our encoding there will always be a range of n numbers, which s must avoid. If s can assume any value from the plaintext space, then the protocol is technically incomplete. The probability of overflow is certainly negligible if s is generated uniformly from the whole range, however if (some or all) clerks can collude to produce overflow problems, then they can alter the outcome of the election.

Coercion As described in [XSHT08], the possibility of coercion arises post-tabulation, when the value $(r + s) \pmod M$ is made public. If corrupt registrars, having secretly recorded a list of all the seed values, could identify s from this value, then they could infer r and thus link the ballot to its result.

As noted in [RS06], this is unlikely to lead to feasible coercion *per se*, because only the registrars learn the value of s, and they have no obvious way to link it to a particular voter because they do not see the teller onion. However a practical attack arises if a target voter can prove which s value was used to generate their ballot.[7] We note that one scenario in which this occurs is if the voter is allowed to see $[\![s]\!]_{Reg}$ on their ballot form. This provides a method of linking a voter to their ballot.

The attack can be summarised in three steps:

1. Corrupt registrars secretly record the association between every onion $[\![s]\!]_{Reg}$ and its decrypted s value in the candidates printing phase.
2. A target voter is then asked to memorize $[\![s]\!]_{Reg}$ on their ballot (even the first few characters would probably identify it).
3. Registrars hence learn the voter's s value, and therefore post-tabulation, their r value (in all likelihood).

One solution would be to attempt to limit the range of s. It is suggested in [RS06, XSHT08] that honest clerks could contribute in a way which results in s taking on $O(n\sqrt{l})$ and $O(nl)$ values respectively. However the problem now is that a single malicious clerk has even more control over s. Apart from frustrating attempts to limit s, they could even mount the overflow attack described earlier with significant probability.

The other obvious solution is to prevent the voter from seeing $[\![s]\!]_{Reg}$ at all, in fact [RS06] suggest that this onion could be destroyed after use. However the cost of physically implementing this in a way that also allows auditing, may well be greater than the alternative we propose below, which never requires $[\![s]\!]_{Reg}$ to be hidden.

4.3 Improved Ballot Construction

We present two distributed ballot constructions, one for single-candidate selection, and one for more expressive voting schemes, which address the incompleteness and coercion issues described above.

Modular Seed Composition A way to avoid the problems of incompleteness and coercion is to use the "partial" homomorphism modulo N, described in Section 2.2. We will choose the group so that the modulus N is equal to the number of candidates, n.

[7] Compare this to the attack of [RT09] surveyed in Appendix C, where a coercer demands to know σ from a voter.

Specifically we instantiate an El Gamal scheme in a group $\langle g \rangle$ of prime order $q \equiv 1 \pmod{n}$ and compute α of order n. We form public keys $PK_{Reg} = (g, h_R)$, $PK_{Tel} = (g, h_T)$. Then we can absorb an index value into the teller onion, since: $E_{Tel}(s) \otimes \alpha^{-r} = E_{Tel}(s-r)$.

Since the onions now encode α^s rather than s, the ballot generation process must change, we show how parallel shuffling solves this.

Verifiable Joint Generation of Ballot Onions The aim is to produce ballots of the form $[(g^{\gamma_1}, g^{\alpha^s} {h_R}^{\gamma_1}), (g^{\gamma_2}, g^{\alpha^s} {h_T}^{\gamma_2})]$, where $s \in_R \mathbb{Z}_n$.

l clerks will generate v ballots (or possibly more in practice to allow auditing). The following ciphertexts with zero-randomness are passed as input to the 1st clerk.

$$\left([[\![\alpha^0]\!]_{Reg}, \ldots, [\![\alpha^{v-1}]\!]_{Reg}], [[\![\alpha^0]\!]_{Tel}, \ldots, [\![\alpha^{v-1}]\!]_{Tel}] \right).$$

When the jth clerk is passed the following pair by the $j-1$th clerk:

$$\begin{array}{ll} (& [[\![\alpha^{\sigma_{j-1}(0)}]\!]_{Reg}, \ldots, [\![\alpha^{\sigma_{j-1}(v-1)}]\!]_{Reg}], \\ & [[\![\alpha^{\sigma_{j-1}(0)}]\!]_{Tel}, \ldots, [\![\alpha^{\sigma_{j-1}(v-1)}]\!]_{Tel}] \quad). \end{array}$$

she thinks of a private permutation $\phi_j \in_R S_n$, and shuffles the lists in parallel, outputting:

$$\begin{array}{ll} (& [[\![\alpha^{\phi_j \sigma_{j-1}(0)}]\!]_{Reg}, \ldots, [\![\alpha^{\phi_j \sigma_{j-1}(v-1)}]\!]_{Reg}], \\ & [[\![\alpha^{\phi_j \sigma_{j-1}(0)}]\!]_{Tel}, \ldots, [\![\alpha^{\phi_j \sigma_{j-1}(v-1)}]\!]_{Tel}] \quad) \end{array}$$

The result after all clerks are finished is:

$$\begin{array}{ll} (& [[\![\alpha^{\sigma_l(0)}]\!]_{Reg}, \ldots, [\![\alpha^{\sigma_l(v-1)}]\!]_{Reg}], \\ & [[\![\alpha^{\sigma_l(0)}]\!]_{Tel}, \ldots, [\![\alpha^{\sigma_l(v-1)}]\!]_{Tel}] \quad) \end{array}$$

where $\sigma_l = \phi_l \ldots \phi_1$.

By "zipping" the two output lists together, and taking the corresponding pairs, v ballots are formed. The parallel mixing process is made verifiable by $Shuf_{Par}$.

The communication complexity of the above process is $O(vl)$, which is the same as [RS06], [Rya08]. Even though the hidden constant may be larger due to the use of mixing, the benefits are:

- The ballot generation process is now complete.
- The decryption of mixed receipts avoids the coercion issue described in Section 4.2.2. If at least one ballot generation clerk and at least one decryption clerk keep their permutations secret, we know of no way the others can infer which ballots correspond to which decrypted votes.
- If a voter is handed a ballot, each cyclic candidate ordering is essentially equiprobable.

Note that the first and last point hold if we use parallel shuffling alone, without the special encoding.

4.3.1 Verifiable Ballot Generation in Ranked Voting

In general ranked voting has not been coercion-resistantly achieved in Prêt à Voter, we refer to Appendix C for a characterisation of the challenge that this problem poses. An exception is [Hea07], however details of how to efficiently verify ballot construction are omitted from that paper. What we describe next is how $Shuf_{Par}$ solves precisely this problem, leading to an efficiently verifiable coercion-resistant ranked-voting scheme.

Candidate Permutation Representation In [Hea07] an encrypted candidate permutation σ is represented via a vector of onions (one onion holding each candidate identifier), note that this approach is also suggested in [Rya08].

Joint Generation of Ballot Onions For convenience, let the candidate identifiers be the integers $1, \ldots, n$. l clerks can jointly generate an encrypted ballot form, using a similar construction to the single-value case described in the previous section. To generate a ballot form, we start off with a pair of vectors:

$$\langle [\![1]\!]_{Reg}, \ldots, [\![n]\!]_{Reg} \rangle - \langle [\![1]\!]_{Tel}, \ldots, [\![n]\!]_{Tel} \rangle$$

For $1 \leq j \leq l$ the jth clerk sequentially applies shuffle ϕ_j in parallel to the above vectors, the result is:

$$\langle [\![\sigma_l(1)]\!]_{Reg}, \ldots, [\![\sigma_l(n)]\!]_{Reg} \rangle - \langle [\![\sigma_l(1)]\!]_{Tel}, \ldots, [\![\sigma_l(n)]\!]_{Tel} \rangle$$

Provided at least one clerk is honest, $\sigma =: \sigma_l^{-1}$ is uniformly chosen from S_n. The left and right vectors form the tellers' and registrars' encryptions of σ respectively. Once again the process is made verifiable by $Shuf_{Par}$.

Anonymous Tabulation Absorbing the plaintext part of the receipt, π_r, into the rest of the receipt is very easy and publicly computable. The vector of onions is simply permuted according to π_r. The new receipts are now passed through a re-encryption mixnet to break the voter-vote correspondence (that is shuffling occurs between receipts; the order of encrypted candidate identifiers within

a receipt is unchanged). This is achievable by parallel shuffling. We first zip the votes together, forming n vectors each of which is v ciphertexts long. These are then shuffled in parallel and the output is zipped. In this case all the ciphertexts are under one public key (that of the Tellers), so we do not strictly require $Shuf_{Par}$ and could instead use the "ElGamal Sequence Shuffle Proof Protocol" in [Nef04].

5 Conclusion

We have defined verifiable parallel shuffling and presented efficient parallelisations of three shuffle zero knowledge proofs. Of these, the parallelisation of [Nef04] is both highly efficient and unconditionally sound, making it ideal for electronic voting. We also reviewed the Prêt à Voter electronic voting scheme, and discussed several problems in existing approaches. We presented a cryptographic primitive that allows modular addition of a encrypted message with a plaintext message. The accompanying ballot generation process ensures a uniform distribution of cyclic candidate orderings, and can be made verifiable by parallel shuffling proofs. The combined process hence yields a coercion resistant scheme for single-value elections. Parallel shuffling is also applicable to verifiable ballot generation in ranked-voting elections. Future directions include finding other applications of the modular homomorphism and of parallel shuffling.

6 Acknowledgements

Thanks to Peter Ryan, Zhe (Joson) Xia and several anonymous reviewers for helpful comments on this work.

References

[BMN+] Josh Benaloh, Tal Moran, Lee Naish, Kim Ramchen, and Vanessa Teague. Shuffle-Sum: Coercion-Resistant Verifiable Tallying for STV Voting. To appear in *IEEE Transactions on Information Forensics & Security, special issue on Electronic Voting*.

[Bon98] Dan Boneh. The Decision Diffie-Hellman Problem. In *Proceedings of the Third Algorithmic Number Theory Symposium, LNCS 1423*, pages 48–63. Springer-Verlag, 1998.

[BT94] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *Symposium on Theory of Computing*, pages 544 – 553. ACM, 1994.

[FS01] Jun Furukawa and Kazue Sako. An Efficient Scheme for Proving a Shuffle. In *Proc. CRYPTO '01, LNCS 2139*, pages 368–387. Springer-Verlag, 2001.

[GL07] Jens Groth and Steve Lu. Verifiable Shuffle of Large Size Ciphertexts. In *Proc. PKC '07, LNCS 4450*, pages 377–392. Springer-Verlag, 2007.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that Yield Nothing but Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *J. ACM*, 38(3):690–728, 1991.

[Gro05] Jens Groth. A Verifiable Secret Shuffle of Homomorphic Encryptions (Updated). `www.brics.dk/~jg/JournalShuffle.pdf`, 2005.

[Hea07] James Heather. Implementing STV securely in Prêt à Voter. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF '07)*, pages 157–169. IEEE Computer Society, 2007.

[HSvV09] Sebastiaan Hoogh, Berry Schoenmakers, Boris Škorić, and José Villegas. Verifiable Rotation of Homomorphic Encryptions. In *Proc. PKC '09, LNCS 5443*, pages 393–410. Springer-Verlag, 2009.

[Nef01] C. Andrew Neff. A Verifiable Secret Shuffle and its Application to E-Voting. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS '01)*, pages 116–125. ACM, 2001.

[Nef04] C. Andrew Neff. Verifiable Mixing (Shuffling) of ElGamal Pairs. Technical report, VOTEHERE, 2004.

[NR97] Moni Naor and Omer Reingold. Number-Theoretic Constructions of Efficient Pseudo-random Functions. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, pages 458–467. IEEE, 1997.

[NSN04] L. Nguyen and R. Safavi-Naini. An Efficient Verifiable Shuffle with Perfect Zero-knowledge Proof System. In *Cryptographic*

Algorithms and their Uses, pages 40–56, 2004.

[NSNK06] L. Nguyen, R. Safavi-Naini, and K. Kurosawa. Verifiable Shuffles: a Formal Model and a Paillier-based Three-round Construction with Provable Security. *International Journal of Information Security*, 5(4):241–255, 2006.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - Eurocrypt '99*, 1999. LNCS 1592.

[RS06] Peter Ryan and Steve Schneider. Prêt à Voter with Re-encryption Mixes. In *Proceedings of the 11th European Symposium on Research in Computer Science (ESORICS '06), LNCS 4189*, pages 313–326. Springer-Verlag, 2006.

[RT09] Peter Ryan and Vanessa Teague. Permutations in Prêt à Voter. In *Proceedings of the Workshop on Trustworthy Elections (WOTE '09)*, 2009.

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

[SK95] K. Sako and J. Kilian. Receipt-Free Mix-Type Voting Scheme – A Practical Solution to the Implementation of a Voting Booth. In *Proc. EUROCRYPT '95, LNCS 921*, pages 393–403. Springer-Verlag, 1995.

[Xia09] Zhe Xia, 2009. pers. comm.

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

[XSHT08] Zhe Xia, Steve Schneider, James Heather, and Jacques Traoré. Analysis, Improvement, and Simplification of Prêt à Voter with Paillier Encryption. In *Proceedings of the Workshop on Trustworthy Elections (WOTE' 08)*, 2008.

A The Decision Diffie Hellman assumption

The DDH assumption is a computational hardness assumption involving distributions related to certain group families. An example of a group family in which the DDH is believed to hold is $Q_{p,q}$: the subgroup of $\mathbb{Z}_p^*$ of order q, where p and $q \mid p-1$ are primes, with $q > p^{1/10}$ [Bon98].

Definition A.1 *Let $\mathbb{G} = \{G_{\mathfrak{p}}\}$ be a set of finite cyclic groups. Let $\mathcal{IG}$ be a randomized algorithm that given a natural number n (in unary), outputs in polynomial time an index $\mathfrak{p}$ and a generator g of a group $G_{\mathfrak{p}} \in \mathbb{G}$. The DDH assumption for $\mathbb{G}$ is that the following distributions are computationally indistinguishable:*

$$D_{DDH} = \langle \mathfrak{p}, g, g^a, g^b, g^{ab} \rangle : \\ \langle \mathfrak{p}, g \rangle \xleftarrow{d} \mathcal{IG}(n), a, b \in_R \{1, \ldots, |G_{\mathfrak{p}}|\}$$

$$D_{ALL} = \langle \mathfrak{p}, g, g^a, g^b, g^c \rangle : \\ \langle \mathfrak{p}, g \rangle \xleftarrow{d} \mathcal{IG}(n), a, b, c \in_R \{1, \ldots, |G_{\mathfrak{p}}|\}$$

This is equivalent to [Bon98], for all probabilistic polynomial time (PPT) algorithms A:

$$|P[A(\mathfrak{p}, g, g^a, g^b, g^{ab}) = \text{“true”}] - P[A(\mathfrak{p}, g, g^a, g^b, g^c) = \text{“true”}]|$$

is a negligible function of n.

Lemma A.2 *Suppose that the distributions:*

$$D_l \triangleq \langle \mathfrak{p}, g, h, (g^{r_1}, h^{r_1}), \ldots, (g^{r_l}, h^{r_l}) \rangle : \\ \langle \mathfrak{p}, g \rangle \xleftarrow{d} \mathcal{IG}(n), h \in_R G_{\mathfrak{p}}, r_i \in_R \{1, \ldots, |G_{\mathfrak{p}}|\}$$

$$R_l \triangleq \langle \mathfrak{p}, g, h, (g^{r_1}, h^{s_1}), \ldots, (g^{r_l}, h^{s_l}) \rangle : \\ \langle \mathfrak{p}, g \rangle \xleftarrow{d} \mathcal{IG}(n), h \in_R G_{\mathfrak{p}}, r_i, s_i \in_R \{1, \ldots, |G_{\mathfrak{p}}|\}$$

are distinguishable with work W_l and probability ϵ. Then D_{DDH} and D_{ALL} are distinguishable with work $W_l + O(l)$ and probability ϵ.

Proof The proof follows from the well-known notion of *random self reducibility* (see e.g. [NR97, Bon98]) and underpins the notion of ciphertext indistinguishability or *IND-CPA* for El Gamal. We note that ciphertexts from D_l, R_l correspond to uniformly drawn ciphertexts encrypting 0, and *any* plaintext respectively.

B El Gamal Ciphertexts

We present a parallelisation of the *ElGamal k-Shuffle Proof Protocol* [Nef04], which being unconditionally

sound, is ideal for use in electronic voting. Since our protocol is very similar and indeed reduces to the ElGamal k-Shuffle Proof Protocol in the single public-key case, we have written this section with careful references to the corresponding sections in [Nef04].

B.1 Introduction

We require a group $\mathcal{G} = \langle g \rangle$ of prime order q, in which the DDH assumption holds. [Nef04] used a subgroup of the multiplicative group $\mathbb{Z}_p^*$, p a large prime. Then $q|p-1$. By enforcing $\gcd(q^2, (p-1)) = q$, there is a unique order q subgroup [Nef04].

Notation We introduce some more notation into the definition of $Shuf_{Par}$ in Section 3.1. Call the J input vectors of ciphertexts: $\{(X_{1,i},\ Y_{1,i})\}, \ldots, \{(X_{J,i},\ Y_{J,i})\}$ and the J output vectors of ciphertexts $\{(\bar{X}_{1,i},\ \bar{Y}_{1,i})\}, \ldots, \{(\bar{X}_{J,i},\ \bar{Y}_{J,i})\}$. These, as well as g and public keys, (g_j, h_j), are all publicly known elements of $\langle g \rangle = \mathcal{G}$.

Then the prover, $\mathcal{P}$, knows $\beta_{j,i} \in \mathbb{Z}_q$ and $\pi \in S_k$ such that for each $1 \leq j \leq J$, for all $1 \leq i \leq k$:

$$(\bar{X}_{j,i},\ \bar{Y}_{j,i}) = (X_{j,\pi(i)},\ Y_{j,\pi(i)}) \cdot ({g_j}^{\beta_{j,\pi(i)}},\ {h_j}^{\beta_{j,\pi(i)}}). \tag{2}$$

$\mathcal{P}$ is required to convince verifier, $\mathcal{V}$, of the existence of $\beta_{j,i}$ and π satisfying the above equations, without revealing any information about them.

B.2 Neff's Shuffle Protocol

For completeness we summarise the ideas behind the ElGamal k-Shuffle Proof Protocol, however the reader is referred to the original paper for the full details. The main mathematical tool is the Schwartz-Zippel lemma, which says that if a nonzero polynomial is evaluated at a randomly chosen point then the probability it evaluates to zero is small.

Lemma B.1 (Schwartz-Zippel) *Let p be a multivariate polynomial of degree d with coefficients from $\mathbb{Z}_q$. Then the probability that $p(x_1, \ldots, x_m) = 0$ for independently chosen $x_1, \ldots, x_m \in_R \mathbb{Z}_q$ is at most $\frac{d}{q}$.*

For $d = 1$, given fixed $(c_1, \ldots, c_m) \neq \vec{0}$ and c:

$$P[c_1 x_1 + \cdots + c_m x_m = c] \leq \frac{1}{q} : x_i \in_R \mathbb{Z}_q$$

Definition B.2 (Simple k-Shuffle [Nef04]) *Suppose $\{X_i\}_{i=1}^k, \{Y_i\}_{i=1}^k, g, \Gamma$ are public elements of $\langle g \rangle = \mathcal{G}$. $\mathcal{P}$ is required to convince $\mathcal{V}$ of the existence of $\gamma \in \mathbb{Z}_q$ and $\pi \in S_k$ such that $\Gamma = g^\gamma$ and for all $1 \leq i \leq k$: $Y_i = X_{\pi(i)}^\gamma$ - without revealing any information about $\log_g X_i, \log_g Y_i, \gamma$ or π.*

Theorem B.3 (Theorem 1 [Nef04]) *The Simple k-Shuffle Proof Protocol[Nef04] is a sound, complete and honest verifier zero knowledge implementation of the Simple k-Shuffle.*

The key idea of the El Gamal k-Shuffle Proof Protocol is to use the Simple k-Shuffle Proof Protocol to prove a commitment to the secret shuffle π, and link the commitment to the El Gamal ciphertexts, the details can be found in [Nef04].

B.3 Parallel Shuffling: Single Key

We recall that in the case that the public keys (g_j, h_j) are the one and the same, then the Schwartz-Zippel lemma leads to a trivial solution to parallel shuffling: the *ElGamal Sequence Shuffle Proof Protocol* [Nef04]. The verifier generates $e_j \in_R \mathbb{Z}_q$. For $1 \leq i \leq k$ the ciphertexts

$$(\hat{X}_i, \hat{Y}_i) = \prod_{j=1}^{J} (X_{j,i}, Y_{j,i})^{e_j}$$

$$(\bar{X}_i, \bar{Y}_i) = \prod_{j=1}^{J} (\bar{X}_{j,i}, \bar{Y}_{j,i})^{e_j}$$

are formed. The ElGamal k-Shuffle Proof Protocol is then run to show the permuted equivalence of $\{(\hat{X}_i, \hat{Y}_i)\}$ and $\{(\bar{X}_i, \bar{Y}_i)\}$.

B.4 Parallel Shuffling: Distinct Keys

The procedure in the previous section is enabled by the fact that homomorphic operators can element-wise combine the ciphertexts from different input vectors into a single ciphertext. However clearly when the public keys are distinct, the protocol itself needs to parallelised, which we call the *Parallel ElGamal k-Shuffle Proof Protocol*.

Before presenting the parallelisation in Section B.4.1 we note the following changes to the ElGamal k-Shuffle Proof Protocol.

- **PEGA**: $2, 3, 4, 6$ are *identical* to [Nef04] and serve to commit to a permutation π_0.

- **PEGA**: $1, 5, 7$ are *parallelised* from [Nef04] to each set of input and output ciphertexts. We briefly describe what these steps in conjunction show, however a fuller explanation is in the original paper. For each $1 \leq j \leq J$ define:

$$\log_{g_j} X_{j,i} =: x_{j,i}, \ \log_{h_j} Y_{j,i} =: y_{j,i}$$
$$\log_{g_j} \bar{X}_{j,i} =: \bar{x}_{j,i}, \ \log_{h_j} \bar{Y}_{j,i} =: \bar{y}_{j,i}$$

for all $1 \leq i \leq k$.
Applying *Lemma 6* [Nef04] there exists $\pi_0 \in S_n$ such that for each $1 \leq j \leq J$:

$$\sum_{i=1}^{k} b_{\pi_0(i)}(\bar{x}_{j,i} - \bar{y}_{j,i}) - \sum_{i=1}^{k} b_i(x_{j,i} - y_{j,i})$$

is a constant, yet $\{b_i\}_{i=1}^k$ are random.

Then by *Lemma 4* [Nef04]: [8]

$$\bar{x}_{j,i} - \bar{y}_{j,i} = x_{j,\pi_0(i)} - y_{j,\pi_0(i)}$$

except with probability atmost $\frac{2}{q}$. This shows precisely that (2) holds except with negligible probability.

The protocol properties derive largely from the original. Both protocols are *permutation hiding* [NSNK06] - see Section B.4.4. The soundness error is linear in the number of input vectors. The parallelisation is efficient, requiring only a few additional messages per additional input vector.

[8] Essentially the Schwartz-Zippel lemma where the polynomial is evaluated on points whose co-ordinates are all different and non-zero.

B.4.1 Parallel ElGamal k-Shuffle Proof Protocol

PEGA.1. Let $\gamma, a_i, u_i, w_i, \tau_j \in_R \mathbb{Z}_q$. $\mathcal{P}$ computes

$$\Gamma = g^{\gamma}$$
$$A_i = g^{a_i}$$
$$C_i = A_{\pi(i)}^{\gamma} = g^{\gamma a_{\pi(i)}}$$
$$U_i = g^{u_i}$$
$$W_i = g^{\gamma w_i}$$
$$M_1 = g_1{}^{\tau_1 + \Sigma_{i=1}^k w_i \beta_{1,\pi(i)}} \prod_{i=1}^{k} X_{1,i}^{w_{\pi^{-1}(i)} - u_i}$$
$$M_1' = h_1{}^{\tau_1 + \Sigma_{i=1}^k w_i \beta_{1,\pi(i)}} \prod_{i=1}^{k} Y_{1,i}^{w_{\pi^{-1}(i)} - u_i}$$
$$\vdots$$
$$M_J = g_J{}^{\tau_J + \Sigma_{i=1}^k w_i \beta_{J,\pi(i)}} \prod_{i=1}^{k} X_{J,i}^{w_{\pi^{-1}(i)} - u_i}$$
$$M_J' = h_J{}^{\tau_J + \Sigma_{i=1}^k w_i \beta_{J,\pi(i)}} \prod_{i=1}^{k} Y_{J,i}^{w_{\pi^{-1}(i)} - u_i}$$

and reveals the ordered sequences A_i, C_i, U_i, W_i and M_j, M_j' along with Γ to $\mathcal{V}$.

PEGA.2. $\mathcal{V}$ chooses $\rho_i \in_R \mathbb{Z}_q$, computes

$$B_i = g^{\rho_i} / U_i$$

and returns ρ_i as a challenge to $\mathcal{P}$.

PEGA.3. $\mathcal{P}$ computes $b_i = \rho_i - u_i$, then computes

$$d_i = \gamma b_{\pi(i)}$$
$$D_i = B_{\pi(i)}^{\gamma} = g^{d_i}$$

and reveals D_i to $\mathcal{V}$.

PEGA.4. $\mathcal{V}$ generates $\lambda \in_R \mathbb{Z}_q$ and returns it to $\mathcal{P}$ as a challenge.

PEGA.5. $\mathcal{P}$ computes σ_i and $\underline{\tau}_j$ where

$$\sigma_i = w_i + b_{\pi(i)}$$
$$\underline{\tau}_1 = -\tau_1 + \sum_{i=1}^{k} b_{\pi(i)} \beta_{1,\pi(i)}$$
$$\vdots$$
$$\underline{\tau}_J = -\tau_J + \sum_{i=1}^{k} b_{\pi(i)} \beta_{J,\pi(i)}$$

and reveals the ordered sequences σ_i, τ_j to $\mathcal{V}$.

PEGA.6. $\mathcal{P}$ and $\mathcal{V}$ execute the Simple k-Shuffle on the tuple

$$(\{A_i {B_i}^{\lambda}\}, \{C_i {D_i}^{\lambda}\}, g, \Gamma).$$

PEGA.7. Finally, $\mathcal{V}$ evaluates:

$$\begin{aligned}
\Phi_1 &= \prod_{i=1}^{k} \bar{X}_{1,i}^{\sigma_i} {X_{1,i}}^{-\rho_i} \\
\Phi'_1 &= \prod_{i=1}^{k} \bar{Y}_{1,i}^{\sigma_i} {Y_{1,i}}^{-\rho_i} \\
&\vdots \\
\Phi_J &= \prod_{i=1}^{k} \bar{X}_{J,i}^{\sigma_i} {X_{J,i}}^{-\rho_i} \\
\Phi'_J &= \prod_{i=1}^{k} \bar{Y}_{J,i}^{\sigma_i} {Y_{J,i}}^{-\rho_i}
\end{aligned}$$

and checks that

$$\begin{aligned}
\Gamma^{\sigma(i)} &= W_i D_i \\
M_1 g_1^{\tau_1} &= \Phi_1 \\
M'_1 h_1^{\tau_1} &= \Phi'_1 \\
&\vdots \\
M_J g_J^{\tau_J} &= \Phi_J \\
M'_J h_J^{\tau_J} &= \Phi'_J
\end{aligned}$$

B.4.2 Completeness

The protocol does not introduce any additional *zero conditions*.[9] Therefore, as discussed in *T2.1* [Nef04], the probability of completion failure is at most $\frac{k(k+2)}{q}$, which is negligible.

B.4.3 Soundness

We follow the original proof of soundness (*T2.3* [Nef04]). *Corollary 6* [Nef04] establishes that the chance there exists *unique* $\pi_0 \in S_k$ such that:

$$c_i = a_{\pi_0(i)} \quad (3)$$
$$d_i = b_{\pi_0(i)} \quad (4)$$

[9]These are conditions that occur with negligible probability but reveal something about π, they are mainly disallowed to make zero knowledge stricter.

holds is at least $1 - \frac{2k+1}{q}$.

Suppose now that such a π_0 exists. For $1 \leq j \leq J$, define:

- the *statement False*$_j$ to be: $\{\exists i, 1 \leq i \leq k : \log_{g_j}(\bar{X}_{j,i}/X_{j,\pi_0(i)}) \neq \log_{h_j}(\bar{Y}_{j,i}/Y_{j,\pi_0(i)})\}$.
- the *event Accept*$_j$ to be: $\{\mathcal{P}$ chooses $\{\rho_i\}$ and λ so that $\mathcal{V}$ accepts $M_j {g_j}^{\tau_j} = \Phi_j$ and $M'_j {h_j}^{\tau_j} = \Phi'_j\}$.

Applying *Lemma 6* [Nef04]:
$False_j \Rightarrow P[Accept_j] \leq \frac{2}{q}$
Suppose there exists $S \subseteq \{1, \ldots, J\}$, $S \neq \emptyset$ such that $False_j$ holds for all $j \in S$, applying the union bound:

$$P[\cup_{j \in S} Accept_j] \leq \textstyle\sum_{j \in S} P[Accept_j] = \frac{2|S|}{q}$$

Forgery occurs if:

- unique π_0 such that (3), (4) hold does not exist.
- A set S described above exists.

Again applying the union bound, the total forgery probability is at most $\frac{2k+1}{q} + \frac{2J}{q} = \frac{2k+2J+1}{q}$

B.4.4 Permutation Hiding

It has been suggested in [NSN04] that zero knowledge of Neff's proof is an open question. However we show that with minor modifications Neff's simulation demonstrates *IND-CPA*$_S$ or *permutation hiding* as defined in [NSNK06]. It is also readily shown that the proof does *not* satisfy the stronger notion of *IND-CTA*$_S$ or *indistinguishability under chosen transcript attacks*, also defined in [NSNK06]. These statements hold identically for the parallelisation and are discussed below.

IND-CPA$_S$ The notion of *IND-CPA*$_S$ is an analogue of ciphertext security under chosen plaintext attack and specifies a game between a shuffle and a PPT adversary as follows:

1. A random instance of a cryptosystem with security parameter n is generated.
2. The adversary selects two permutations $\pi_{(1)}, \pi_{(2)}$ and the input; plaintexts, $L_{in}^{(p)}$, encryption factors, $C_{E_{pk}}^{L_{in}^{(p)}, L_{in}}$ and ciphertexts, L_{in}, to the shuffle.
3. The honest shuffle selects a permutation, $\pi_{(i)}$, randomly and outputs shuffled ciphertexts, L_{out}, and a proof transcript, $VIEW_{\mathcal{V}}^{\mathcal{P}}(pk, L_{in}, L_{out})$.

4. The adversary has negligible (in n) chance greater than $\frac{1}{2}$ of guessing which permutation was used using all the available information.

Equivalently, the following two challenges are computationally indistinguishable

$$o^{\pi_{(1)}} \leftarrow (L_{in}, L_{in}^{(p)}, C_{E_{pk}}^{L_{in}^{(p)}}, L_{out}, VIEW_{\mathcal{V}}^{\mathcal{P}}(pk, L_{in}, L_{out})) :$$

$$L_{out} = L_{in} \text{ permuted by } \pi_{(1)}$$

$$o^{\pi_{(2)}} \leftarrow (L_{in}, L_{in}^{(p)}, C_{E_{pk}}^{L_{in}^{(p)}}, L_{out}, VIEW_{\mathcal{V}}^{\mathcal{P}}(pk, L_{in}, L_{out})) :$$

$$L_{out} = L_{in} \text{ permuted by } \pi_{(2)}$$

This leads to a common technique to prove *IND-CPA*$_S$, namely it suffices to prove the computational indistinguishability of the challenges

$$o^{\pi_{(1)}} \leftarrow (L_{in}, L_{in}^{(p)}, C_{E_{pk}}^{L_{in}^{(p)}}, L_{out}, VIEW_{\mathcal{V}}^{\mathcal{P}}(pk, L_{in}, L_{out})) :$$

$$L_{out} = L_{in} \text{ permuted by } \pi_{(1)}$$

$$o^{g} \leftarrow (L_{in}, L_{in}^{(p)}, C_{E_{pk}}^{L_{in}^{(p)}}, L_{out}, VIEW_{\mathcal{V}}^{\mathcal{P}}(pk, L_{in}, L_{out})) :$$

$$L_{out} = \text{uniformly random ciphertexts}$$

One can then argue that by symmetry, $o^{\pi_{(2)}}$ and o^{g} are computationally indistinguishable. Thus it follows $o^{\pi_{(1)}}, o^{\pi_{(2)}}$ are computationally indistinguishable.

We apply this technique to explain why Neff's argument shows *IND-CPA*$_S$.

Definition B.4 *Let $SS_k(\mathcal{T})$ be the result of running the Simple k-Shuffle simulator (T1.4[Nef04]) on a tuple $\mathcal{T}$.*

Clearly if $\mathcal{T}$ is of the form $(X_i, X_{\pi(i)}^{\gamma}, g, g^{\gamma}) : X_i \in \mathcal{G}, \gamma \in \mathbb{Z}_q$, then by honest verifier zero knowledge the distribution produced by the simulator is identical to that produced by a real prover and honest verifier running the Simple k-Shuffle. If $\mathcal{T}$ is not of that form, the simulator will produce some other distribution.

Theorem B.5 *Suppose that the challenges:*

$$o^{\pi_{(1)}} \leftarrow (L_{in}, L_{in}^{(p)}, C_{E_{pk}}^{L_{in}^{(p)}}, L_{out},$$
$$\Gamma = g^{\gamma}, A_i, B_i, C_i = A_{\pi_{(1)}(i)}^{\gamma}, D_i = B_{\pi_{(1)}(i)}^{\gamma},$$
$$\sigma_i, \rho_i, \underline{\tau}_j, \lambda, U_i = \frac{g^{\rho_i}}{B_i}, W_i = \frac{\Gamma^{\rho_i}}{D_i},$$
$$SS_k(A_i B_i^{\lambda}, C_i D_i^{\lambda}, g, \Gamma),$$
$$M_j = \frac{\prod_{i=1}^{k} \bar{X}_{j,i}^{\sigma_i} X_{j,i}^{-\rho_i}}{g_j^{\underline{\tau}_j}}, M_j' = \frac{\prod_{i=1}^{k} \bar{Y}_{j,i}^{\sigma_i} Y_{j,i}^{-\rho_i}}{h_j^{\underline{\tau}_j}}) :$$
$$L_{out} = L_{in} \text{ permuted by } \pi_{(1)},$$
$$A_i, B_i \in_R \mathcal{G}, \gamma, \sigma_i, \rho_i, \underline{\tau}_j, \lambda \in_R \mathbb{Z}_q$$
$$o^{g} \leftarrow (L_{in}, L_{in}^{(p)}, C_{E_{pk}}^{L_{in}^{(p)}}, L_{out},$$
$$\Gamma = g^{\gamma}, A_i, B_i, C_i, D_i,$$
$$\sigma_i, \rho_i, \underline{\tau}_j, \lambda, U_i = \frac{g^{\rho_i}}{B_i}, W_i = \frac{\Gamma^{\rho_i}}{D_i},$$
$$SS_k(A_i B_i^{\lambda}, C_i D_i^{\lambda}, g, \Gamma),$$
$$M_j = \frac{\prod_{i=1}^{k} \bar{X}_{j,i}^{\sigma_i} X_{j,i}^{-\rho_i}}{g_j^{\underline{\tau}_j}}, M_j' = \frac{\prod_{i=1}^{k} \bar{Y}_{j,i}^{\sigma_i} Y_{j,i}^{-\rho_i}}{h_j^{\underline{\tau}_j}}) :$$
$$L_{out} = \text{uniformly random ciphertexts},$$
$$A_i, B_i, C_i, D_i \in_R \mathcal{G}, \gamma, \sigma_i, \rho_i, \underline{\tau}_j, \lambda \in_R \mathbb{Z}_q$$

can be distinguished with work W and probability ϵ. Then $D_{k(J+2)}$ and $R_{k(J+2)}$ defined in Appendix A, Lemma A.2, can be distinguished with work $W + O(kJ)$ and probability ϵ.

Proof We argue that using only $L_{in}, L_{in}^{(p)}, C_{E_{pk}}^{L_{in}^{(p)}}$ and operations on sets of kJ and $2k$ ciphertexts the adversary will output a transcript. If the ciphertexts are drawn from $D_{k(J+2)}$, the algorithm will output $o^{\pi_{(1)}}$, while if they are drawn from $R_{k(J+2)}$), the algorithm will output o^{g}.

1. The first operation is that the kJ input ciphertexts L_{in} after being permuted, are either multiplied by kJ ciphertexts from $D_{k(J+2)}$ or $R_{k(J+2)}$. In the former case the shuffle is valid, while in the latter, only uniformly random ciphertexts result (revealing no information about $\pi_{(1)}$).

2. The second operation is *T3.3-4* [Nef04], which we reproduce here. Given $2k$ ciphertexts

$\{(\phi_{1i}, \psi_{1i})\}_{i=1}^{k}, \{(\phi_{2i}, \psi_{2i})\}_{i=1}^{k}$, set:

$$\eta \in_R \mathbb{Z}_q$$
$$(A_i, C_i) = (\phi_{1i}, \psi^{\eta}_{1\pi_{(1)}(i)})$$
$$(B_i, D_i) = (\phi_{2i}, \psi^{\eta}_{2\pi_{(1)}(i)})$$
$$\Gamma = h^{\eta}$$

Clearly $\gamma = \eta \log_g h$ is unknown, however it is uniform over $\mathbb{Z}_q$. It is easy to see if the $2k$ ciphertexts are from $D_{k(J+2)}$, then the distribution of $(A_i, B_i, C_i, D_i, \Gamma)$ is that required for $o^{\pi_{(1)}}$, while otherwise it is uniform over $\mathcal{G}^{4k+1}$, which is that required for o^g.

The Simple k-Shuffle simulator is then run on the input $(A_i B_i^{\lambda}, C_i D_i^{\lambda}, g, \Gamma)$, where $\lambda \in_R \mathbb{Z}_q$. By Definition B.4 this produces the correct transcript for either $o^{\pi_{(1)}}$ or o^g.

3. All that remains is to choose $\sigma_i, \rho_i, \underline{\tau}_j \in_R \mathbb{Z}_q$ and compute U_i, W_i, M_j, M'_j. ∎

Lemma A.2 and Theorem B.5 prove that if $o^{\pi_{(1)}}$ and o^g are distinguishable with work W and probability ϵ, then D_{DDH} and D_{ALL} are distinguishable with work $W + O(kJ)$ and probability ϵ. If the DDH assumption holds then the Parallel ElGamal k-Shuffle Proof Protocol is *IND-CPA*$_S$ - taking $J = 1$ implies the original is also.

IND-CTA$_S$ The notion of *IND-CTA*$_S$ is an analogue of ciphertext security against chosen ciphertext attacks. Here, after the challenge is generated by the shuffle, an adversary is allowed access to an oracle which returns the permutation of any valid transcript, excluding the challenge itself.

It is easily seen that neither the original nor the parallelisation satisfy *IND-CTA*$_S$ since, e.g., the modifications

$$M_1 \leftarrow g_1 M_1,$$
$$M'_1 \leftarrow h_1 M'_1,$$
$$\underline{\tau}_1 \leftarrow \underline{\tau}_1 - 1$$

lead to a *different* valid transcript that can be passed to the oracle - revealing the permutation of the challenge.

C Analysis of Ranked Voting in Prêt à Voter

The first pre-requisite for coercion-resistant ranked voting is the use of arbitrary candidate permutations. Recall that a coercer can demand a particular preference permutation μ from a voter. Since a coercer may obtain a voter's index permutation π_r from their receipt, the coercer must be convinced that the candidate permutation that appeared on the voter's ballot could have been ${\pi_r}^{-1}\mu$. The number of possible μ that a coercer may choose from is typically very large (e.g. a non-negligible fraction of $n!$), so the number of candidate permutations that could have appeared on the voter's ballot form, must be correspondingly as large to prevent coercion.

An attack based on a similar idea appears in [RT09] - there a coercer demands to know the candidate permutation that appeared on a voter's ballot just after they have left the booth, but before public tabulation begins.[10] If tabulation later reveals too much about which candidate permutations actually appeared on ballots, the coercer may use this to check the voter's obedience.

Suppose p_{co} is the proportion of permutations that the coercer may choose from, and suppose p_{sus} is the threshold probability of a voter's obedience, above which a coercer will give a voter the benefit of the doubt. To achieve coercion-resistance:

1. A ranked voting scheme must allow at least $p_{co}p_{sus}n!$ possible candidate permutations.
2. A voting scheme, single-value or ranked, in the anonymous tabulation phase, must maintain that a proportion of at least p_{sus} of all possible candidate permutations, could have appeared on actual ballots.

[RS06] only allows candidate permutations to be chosen that are cyclic shifts. Aside from the attacks described in Section 4.2.2, (2) is satisfied, however naturally (1) is not.

[Rya08] proposed a method which allows arbitrary candidate permutations to be chosen, so (1) is satisfied. However that method suffers from the attack already described, so that (2) is not satisfied. The attack results in a proportion of only $\frac{vn}{n!} = \frac{v}{(n-1)!}$ of possible candidate permutations plausibly appearing on ballots. Unless n is very small, this is almost certainly smaller then p_{sus}.

By reducing the space of possible permutations, [RT09] has shown how to satisfy (2). However they note that this breaks (1), so the scheme is suitable for single-value elections but not for the ranked voting case.

[10] Once tabulation has occurred it is easy for the voter to retrospectively pick a plausible candidate permutation.