Ballot Permutations in Prêt à Voter

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

June 23, 2009

Abstract

Handling full permutations of the candidate list along with re-encryption mixes is rather difficult in Prêt à Voter but handling cyclic shifts is straightforward. One of the versions of Prêt à Voter that uses Paillier encryption allows general permutations of candidates on the ballot, rather than just cyclic shifts. This improves the robustness of the system against an adversary who tries to alter checkmarks on ballots before they are posted to the bulletin board. Even if the adversary could predict which voters would fail to check their vote on the bulletin board, the best they could do would be to choose another random candidate. By contrast, when using only cyclic shifts the adversary can systematically shift a biased distribution from one candidate to another.

We show in this note that the Paillier version of Prêt à Voter with full permutations of the candidates is not receipt-free when the number of possible permutations is much larger than the number of voters, and we propose a construction that addresses this issue while retaining the defence against an adversary who can shift checkmarks.

1 Introduction

The key idea behind the Prêt à Voter suite of voter-verifiable schemes is to encode the vote in a *protected ballot* by independently permuting the order of the candidates for each ballot form [Rya04, CRS05]. A cryptographic value printed on the ballot form carries information defining the order shown on the form, see Figure 1. In the booth, the voter marks her X or ranking *etc* against her candidate(s) of choice, then detaches and shreds the candidate order. The resulting form constitutes the ballot receipt, see Figure 2. Without the candidate list or the relevant keys to recover the candidate list, it is not possible to determine what vote is encoded in the receipt, so proving ballot secrecy. After the election, verifiable, anonymising tabulation processes allow the set of original votes to be recovered and tallied. The receipts allow the voters to confirm, typically via a public bulletin board, that their votes are input into the tabulation.

Candidates	Your vote
Obelix	
Idefix	
Asterix	
Panoramix	
	$7rJ94K$
Destroy	Retain

Figure 1: Prêt à Voter ballot form

The original versions of Prêt à Voter used RSA encryption and decryption mixes. This had the advantage that it was straightforward to handle full permutations of the candidates and perform the tabulation without revealing any information about the permutations that appeared on ballots that had been cast. Decryption mixes have the disadvantage of being rather inflexible and difficult to recover in the

Your Vote
X
$7rJ94K$
Retain

Figure 2: Prêt à Voter ballot receipt encoding a vote for "Idefix"

event of errors or corruption being detected. This prompted later versions to incorporate randomising algorithms and re-encryption mixes. These are more flexible and robust but have the disadvantage that it is harder to deal with full permutations, at least in a way that does not reveal information about permutations that appear on cast ballots. It is however quite easy to handle cyclic shifts of the candidate list.

1.1 Shift Happens

While cyclic shifts are clearly enough to hide the voters choice, at least where the voter is required to select just one candidate, there remains a concern regarding the robustness of such an approach. The main concern with cyclic shifts is that if a corrupt poll-worker (or scanner) could predict which voters would not check their vote on the bulletin board, they could move those voters' checkmarks before posting. If ballot permutations were all cyclic shifts of a canonical permutation, then this would provide a systematic way to shift votes from one candidate to another. Suppose that the adversary knows that a majority of votes will be cast for candidate A and she wishes to swing the election in favour of candidate C. She simply applies a cyclic shift of 2 to the position of the X on a suitable proportion of the receipts. Of course, voters or their proxies checking their receipts should catch such manipulation, but it is not clear to what extent we can rely on this.

This is the only known method of vote manipulation against Prêt à Voter 05 [CRS05] that relies on the individual voter's vigilance for its detection. Others, such as incorrect mixing or faulty ballot production, will be detected by auditors or bulletin board observers with some fixed probability. Indeed, shifting checkmarks could be a practical attack if performed by a pollworker who could confidently predict that a certain voter would not verify their recipt on the bulletin board. So, in the spirit of deploying a strategy of layered defence, it is desirable to avoid this possibility.

One of the variants of Prêt à Voter [Rya08], uses Paillier encryption and incorporates an additional cryptographic value on the ballot forms to allow general permutations of candidates along with re-encryption mixes, rather than just cyclic shifts. Unfortunately, the scheme of [Rya08] is not perfectly receipt free. Any permutation can appear on the ballot, and the whole permutation is revealed after mixing, in order to compute the selected candidate. This opens the scheme up to a pattern identification attack (similar to the "Italian attack" on more complex voting methods) even though the voter only selects one candidate.

The purpose of this note is to examine the choice of possible permutations that can appear on ballots. We observe that allowing all possible permutations as values introduces a coercion attack when the number of possible permutations is large. We propose instead a much smaller set of possible permutations, which retains most of the advantages of general permutations over cyclic shifts. Our construction is adequate for first-past-the post elections, but not for more expressive voting schemes such as STV or approval voting.

Our construction retains the advantages of Prêt à Voter with Paillier over earlier versions of Prêt à Voter [CRS05, RS06]. The comparison with Xia *et al*'s modification of Prêt à Voter with Paillier encryption [XSHT08] is more mixed: our construction retains the classic voter experience of simply making a checkmark on a piece of paper, rather than detaching and resticking a scratch strip. However, we neither solve the coerced-randomization attack nor accommodate complex voting schemes.

The following section describes the attack on the original version of Prêt à Voter with Paillier encryption. Section 3 describes our alternative construction for prime numbers of candidates, which is generalized to composites in Section 4. Then Section 5.1 contains a discussion of stronger attack models.

2 Why the scheme is not receipt free when any permutation can appear on the ballot

The permutation of the candidate names that is printed on the ballot in [Rya08] is the composition of a permutation ϕ with a cyclic shift κ. The permutation ϕ is generated by applying some publicly known function f to a random seed value ρ. When a voter selects the candidate at index ι, this value is added to the cyclic shift and then the encrypted pair $(\rho, \iota + \kappa)$ is posted on the bulletin board, mixed with other votes, and finally decrypted. The selected candidate can then be recovered as the $(\iota+\kappa)$-th element of ϕ.

Let n be the number of candidates. Suppose that n is large enough that $n!$ is much larger than the number of voters. (A dozen candidates is enough.) Also suppose that f is a total function over the set of possible permutations, with a close-to-uniform distribution, so that any permutation is roughly equally likely to appear on the ballot. This section describes how a coercer can discover how a person voted, even if the person votes privately in a polling booth.

1. After the voter leaves the polling station, but before the votes are decrypted on the bulletin board, the coercer demands to know the candidate permutation P that was on the ballot. (Of course the voter could lie at this point, but we will argue that he would likely get caught.) The coercer must of course tell the voter beforehand that he will be expected to record the permutation.

2. After tallying, the coercer searches the decrypted values of ϕ on the bulletin board for P or any cyclic shift of it. If she finds no such value, she punishes the voter (who must have lied at Step 1). If she finds only one such value, then she uses the decrypted corrected index to find out how the voter voted.

If the voter lies in Step 1, and chooses a random permutation instead, then only n out of the $n!$ possible permutations will be consistent with that lie. If the number of votes on the bulletin board is much smaller than $(n-1)!$ then there's only a small chance that in Step 2 the coercer will find a permutation consistent with the voter's lie. (And likewise, if the voter tells the truth, the probability of the coercer finding more than one consistent permutation is small.)

If the voter lies in Step 1, and tells the coercer a cyclic shift of P, then he might as well have told the truth—the coercer can still check for the presence of that permutation and discover how he voted.

Of course, if the voter knows of another valid permutation that is guaranteed to be on the bulletin board, then he can tell that to the coercer instead. But this doesn't seem reasonable for most voters, especially since the coercer is probably demanding a vote for a particular candidate, so the coercion-avoiding voter would have to know of an(other) obedient vote.

This shows that the construction is not receipt-free when f ranges over the whole set of permutations. The rest of the paper shows how to restrict the range of f so as to retain both receipt freeness and defence against an attacker who can undetectably shift checkmarks. We begin by describing a construction that works when the number of candidates is prime, then, in Section 4, we generalize this to composites.

3 Restoring receipt-freeness

3.1 Background

The problem described in the previous section arises because the set of possible permutations is typically much larger than the number of voters. The obvious response is to restrict the possibilities to a smaller subset of the permutations, but we need to do this without re-introducing the problems associated with cyclic shifts.

We describe now a construction that ensures that an attacker who tries to alter votes by shifting the checkmark (without decrypting the onion) should not be able to do better than randomise votes. In fact our construction goes further: it allows the tabulation to be performed without having to reveal the set of permutations that appeared on cast ballots. We will discuss counters to attempts to manipulate the receipts in other ways, *i.e.* manipulating the onion values, in Section 5.1.

Number the candidates from 0 to $n-1$. *For this section we assume that n is prime.* (We show how to remove this restriction in Section 4.) Construct an $n-1 \times n$ table T of candidate names, in which the item in the r-th row and j-th column is:

$$T_{rj} = r{\cdot}j \bmod n \text{ for } 1 \leq r \leq n-1 \text{ and } 0 \leq j \leq n-1.$$

An example for $n = 7$ is shown in Figure 3.

For prime n, this construction produces a *circular florentine square* of order n [CD07], defined as follows:

Definition 1. *A* circular florentine square *of order n has $n-1$ rows and n columns such that*

1. each row is a permutation of the n symbols, and

0	1	2	3	4	5	6
0	2	4	6	1	3	5
0	3	6	2	5	1	4
0	4	1	5	2	6	3
0	5	3	1	6	4	2
0	6	5	4	3	2	1

Figure 3: A circular florentine square of order 7.

2. *for any two distinct symbols a and b and for each m from 1 to $n-1$, there is exactly one row in which b is m steps to the right of a.*

The term "circular" in the definition refers to the idea of each row having the end glued to the beginning, so that the first element is one step to the right of the last. This means that moving m steps to the right is the same as moving $n-m$ steps to the left.

Writing a randomly-chosen row of the circular florentine square directly onto the ballot would not quite achieve the security properties we require. The first column is always 0, so receipts would leak information about the vote, and an adversary who wanted to bias the outcome towards or away from candidate 0 could easily do so. More subtly, if the adversary could guess how someone voted, then they could infer the exact position of any other candidate given the position of the checkmark. Returning to the example in Figure 3, if the adversary believed that the voter chose candidate 1, and wanted to shift the vote to candidate 2, then they could shift the checkmark from $j = 1$ to $j = 2$ (row 1), or $j = 4$ to $j = 1$ (row 2), or $j = 5$ to $j = 3$ (row 3), *etc.*

These problems are easily fixed by introducing a random cyclic shift σ to each row of the florentine square. Obviously this implies that a given checkmark position is equally likely to indicate any candidate (including 0). We will show that this solves the second issue as well—any other candidate is equally likely to be located any any other position, even given the checkmark's location and knowledge of which candidate was selected. The properties we need are defined precisely as follows:

Definition 2. *A* pairwise-symmetrical *ballot construction is one in which*

1. *Each candidate's ballot location is uniformly distributed, and*

2. *For all (distinct) candidates A and B, for all ballot positions j, candidate b's location is uniformly distributed (over locations other than j), given that A is located at j.*

This is almost the same as a pairwise-independent set of uniform probability distributions, except that two candidates can't occupy the same location on the ballot. This is sufficient for single-choice elections, but not for more expressive voting methods. For example, if we were to allow voters to select their two favourite candidates, then we would need to generalise the idea to a three-wise symmetrical construction; if we wanted to allow full permutations, then we would need n-wise symmetry, which would take us back to the whole set of permutations.

3.2 Ballot construction

Notation For any integers α and β, $\alpha < \beta$, let $[\alpha, \beta]$ denote the set of integers between α and β, inclusive.

Notation For any variable x and finite set S, write $x \in_R S$ to mean that x is chosen uniformly at random from S.

3.2.1 The main idea: *perfectly shifted florentine square*

Index the locations on the ballot form from 0 to $n-1$, listed downwards. The ballot order is defined by a row $k \in_R [1, n-1]$, and a cyclic shift $\sigma \in_R [0, n-1]$. The candidate in the i-th location is determined by computing $i \cdot k + \sigma \bmod n$. Equivalently, the list of candidates on the ballot is derived by selecting the k-th row of the florentine square of order n, then adding σ to each value, mod n.

Lemma 1. *A perfectly-shifted florentine square is a pairwise-symmetrical ballot construction.*

Proof. We consider the two parts of Definition 2 in turn.

1. The uniformly distributed ballot location follows from σ being uniformly chosen in $[0, n-1]$, and each row of the florentine square containing each number exactly once.

2. Suppose candidate A is at location j, and consider the location i of candidate $B \neq A$. For each pair (j, i) with $i \neq j$ there is exactly one pair of ballot parameters (k, σ) that would place A at location j and B at location i. (To prove this, suppose instead that there are two pairs

(k, σ) and (k', σ') such that $A = ik + \sigma \bmod n$ and $A = ik' + \sigma' \bmod n$ and $B = jk + \sigma \bmod n$ and $B = jk' + \sigma' \bmod n$ with $i \neq j$. Since n is prime, the only solution is $\sigma = \sigma'$ and $k = k'$.) Since all pairs (k, σ) are equally likely *a priori*, each value of j is equally likely.

□

3.2.2 Practical details - the *almost-perfectly shifted florentine square*

In practice, instead of using $\sigma \in_R [0, n-1]$ we would instead choose $s \in_R \mathbb{Z}_q^*$, and take the final result mod n. (q is the size of the domain of the encryption function (El Gamal or Paillier), with length proportional to the security parameter.) This is important for perserving privacy during tabulation, as described below. This would produce a distribution of negligible statistical distance from a pairwise-symmetrical one. We call this the *almost-perfectly shifted florentine square construction.* It is summarised in Protocol 1.

Protocol 1 Almost-perfectly shifted florentine square construction

```
1: Each ballot form comes equipped with two onions, $\Theta_k$ and $\Theta_s$.
2: $\Theta_k$ is the encryption using exponential ElGamal or Paillier of $k$, where $k \in_R [1, n-1]$ and can be thought of as indexing the row of the florentine square.
3: $\Theta_s$ is the encryption of $s \in_R Z_q^*$ where $s$ determines a cyclic shift mod $n$.
4: Location $i$ corresponds to candidate $i \cdot k + s \bmod n$.
```

Corollary 1. *The* almost-perfectly shifted florentine square construction *produces a distribution of negligible distance from a pairwise-symmetrical one.*

Proof. The statistical distance between the distributions produced by the almost-perfectly shifted florentine square construction and the perfectly shifted one is at most the difference between the two distributions

- $\hat{s}$ uniformly chosen from $[0, n-1]$
- s, uniformly chosen from Z_q^* and reduced mod n

The difference is at most $(n-1)/q$, which for q the order of a typical ElGamal group will be negligible. □

3.3 Tabulation and Decryption

Notation $\{m\}_{PK}$ denotes the encryption of message m under public key PK.

We assume that that all onions are formed using either exponential El Gamal or Paillier, and so enjoy the homomorphic property:

$$\{x\}_{PK} \cdot \{y\}_{PK} = \{x + y\}_{PK}$$

Suppose that the voter chooses the ith cell on such a ballot. We see that they will have selected the $i \cdot k + s \bmod n$ candidate. The receipt will have the form: (i, Θ_k, Θ_s), so this can be transformed into a single ciphertext term $\widehat{\Theta}$ suitable for the mix by computing:

$$\widehat{\Theta} := (\Theta_k)^i \cdot \Theta_s$$

Which will equal $\{k \cdot i + s\}_{PK}$, *i.e.* an encryption of the voter's choice. Note that this transformation can be performed on the Bulletin board and is universally verifiable. Strictly speaking there is a faint possibility of the modular arithmetic of the cryptographic algorithm interfering with the mod$n$ arithmetic. However, $k \cdot i$ is bounded by n^2 so problems would only arise if $q - s < n^2$ where q is the order of the El Gamal group for example. Clearly if s is chosen at random from Z_q^* the chance of this is vanishingly small.

We can send such terms through a mix and then decrypt them and take the results mod n to recover the voter's choices. Notice that the presence of the s term serves to mask any information about the k and i terms that might have been revealed had the product $i \cdot k$ been revealed. This is important as we do not want our construction to allow the adversary to deduce anything about the linkage between decrypted terms and receipts. It is particularly important to conceal the i value as this would indicate the subset of receipts with the X in this position.

3.4 Receipt freeness

A formal proof of receipt freeness is beyond the scope of this paper. We provide here an informal explanation of why our proposed scheme does not suffer from the coercion issue described in Section 2. The main idea in coercion resistance definitions including [MN06], [PR06] and [DKR05] is that a coerced voter has a stragegy (called a "coercion resistance strategy" in [MN06]) that allows him to pretend to cooperate with the coercer but vote instead for the candidate of his choice. This is successful if the coercer cannot tell whether the voter cooperated or disobeyed.

Suppose a coercer attempts the attack described in Section 2, demanding a particular candidate, and asking to know the ballot permutation as proof that that candidate was selected. The voter instead chooses a different candidate. The voter has a simple coercion resistance strategy: he simply chooses any valid row of a florentine square, rotates it so that his checkmark is consistent with the vote the coercer demanded, then claims that that was the ballot permutation. One easy way to find a valid row is to use the one that really was printed on the ballot, appropriately rotated. This sort of resistance strategy should be simple enough for almost all voters. This is a successful coercion resistance strategy because no step of the public tallying process contradicts this claim. The coercer cannot distinguish the truth from a lie based on the ballot onion (which she cannot decrypt), and she cannot identify the coerced voter's contribution from the decrypted output (assuming that at least one voter chose the candidate she demanded, and that at least one of the mix servers keeps their shuffle secret).

Depending on exactly how ballots are produced and distributed, this construction may still suffer from the privacy problem mentioned by Xia *et al* [XSHT08], Sec 4.3(7). Because s is chosen randomly from a very large range, a particular value of s is probably consistent with only one of the $k \cdot i + s$ values decrypted on the bulletin board. Whoever decrypts s to produce the ballot permutation can infer from the bulletin board, with high probability, what vote was cast with that ballot. The mitigations suggested by Xia *et al* would work here too, for example choosing s from a smaller range (taking care that that did not introduce another privacy violation), or ensuring that the ballot producer did not know the identity of the voter who used the ballot. Alternatively, it is possible to design a protocol in which the authorities could reveal only $k \cdot i + s \bmod n$.

3.5 Set-up

We now describe how one can create the ballots in a distributed fashion in such a way as to ensure that no single entity can undermine privacy by leaking information. Our construction is also designed to counter *kleptographic* attacks, [**?**], in which an entity that determines cryptographic variables can exploit this capability to establish subliminal channels.

Firstly we need to generate a sufficient number of Θ_k onions with the k values evenly distributed across $[1, n-1]$. A simple way to achieve this is to firstly generate a sufficient batch of k values, evenly distributed, and publicly encrypt these with Exponential ElGamal or Paillier. These are then put through a number of verifiable re-encryption mixes by a set of *Clerks*. The result is a batch of k onions of the correct form with the correct distribution.

For Paillier encryption, the s onions require no special constraint on the plaintexts other than the values be randomly selected from Z_q^*. For this we can use the style of construction of [Rya08] to ensure that several entities contribute to the randomness of the s values, so eliminating the possibility of kleptographic channels. In essence, onions are constructed as the product of onions supplied by a number of ballot generation authorities. The final onions therefore contain entropy supplied by several sources and all would have to collude to determine the values in the final onions.

For Exponential ElGamal we need to constrain the s values to render the decryption feasible. We might constrain the values to be drawn at random from a suitable binomial distribution along the lines suggested in [RS06].

The k and s onions so generated can be paired off at random to form the encrypted proto-ballots. Once paired off, the onion pairs are digitally signed by the Voting Authority.

We need now to consider how best to reveal the candidate order. We might do this ahead of time or on-demand when the voter is ready to cast her vote. In either case, for each ballot we need to send the pair of onions to a threshold set of Tellers to reveal the k and s values so allowing the candidate order to be derived and printed on the ballot. Doing this in advance allows for ballots to be audited for well-formedness in advance. If we do this on-demand in the booth we need to introduce some form of cut-and-choose style protocol: for example, allow the voter to choose a set of encrypted ballots, have them all decrypted and the candidate orders committed in print, then auditing all but one, chosen at random. Printing the ballots in advance allows for a simpler voter experience but introduces chain of custody problems and involved greater reliance on auditing authorities. On-demand avoids chain of custody and empowers voters to trigger the ballot audits but makes for a more complex voting ceremony. For a full discussion of these options and the trade-offs between them see

[Rya07].

3.6 Summary

The crucial property of the florentine square is that for any given, distinct candidates A and B, and any given shift $\delta \in [1, n-1]$ there is exactly one row of the square such that A and B are cyclically separated by a shift of δ. Consequently, the adversary cannot systematically shift a distribution centered on one candidate onto another, at least not by shifting the index value in the receipts. Given a ballot, any shift to the position of the X will be equally likely to produce any other candidate. The adversary can of course just seek to randomise (a subset of) the votes but this is a much weaker attack. At worst, by altering a significant proportion of the ballots, the adversary can shift he outcome towards a draw.

Our construction avoids coercion attacks as the tabulation process reveals nothing about the permutation on the ballot form. Our construction also helps with what we might call *2-approval voting*, in which voters are allows to cast up to two votes, in that the resulting receipts are fully hiding: reveal nothing about the voters choice. Unfortunately the construction does not fully eliminate bias attacks.

This completes the scheme in the case that n is prime. We now show how to extend it to a composite number of candidates.

4 When the number of candidates is not prime

If n is not prime, the florentine square construction above does not work. We could simply add some extra dummy candidates to the list until the total is prime. This has the feature that voters could effectively spoil their ballot by selecting a "NULL" candidate rather than a real one. Indeed, if there are several "NULL" options, it allows possibilities for voters to express distinctions between the various flavours of "NULL" vote: "I don't wish to vote for any of these characters" to "Any of these is ok with me" to "I'm only putting a X on this bloody form because I get fined if I don't". Most likely the politicians will be disinclined to allow voters to express their feelings so eloquently.

The number of dummies typically would not be very large—there would never be a need to add more than 5 when there were fewer than 90 candidates. However, it could be argued that this changes the voters' experience too much. Hence we provide an alternative construction which allows the authorities to print an arbitrary number of candidate names on the ballot, while preserving the same symmetry properties as in the circular florentine square.

4.1 Venetian rectangles

We will describe how to generate a pairwise-symmetrical ballot construction for composite numbers of candidates. Let p be the largest prime number less than n, and let $m = n - p$. Remember that m is very small, unlikely ever to be more than 5. We will use the florentine square of order p and add m extra symbols, then rearrange things so that the symmetry still holds.

In the next section we will describe what onions are written on the ballot—for now, we simply consider how to generate the ballot permutation. We call the result a *venetian rectangle*. The construction is summarised in Protocol 2.

We describe a recursive construction which, starting with a florentine square for a prime p, allows us to construct a rectangle for n elements with the appropriate balance property.

Construction: Start with a florentine square F_p of (prime) order p, and define the V_p to be equal to F_p. This has $p-1$ rows. To move to the next level of the hierarchy and obtain the venetian rectangle V_{p+k+1} from V_{p+k}, we proceed as follows: we add the numeral $p+k$ in all possible gaps between the elements of the V_{p+k}. Due to the circular nature of the rows, adding a symbol at the beginning is the same as adding it at the end.

Terminology: We define the following predicate on rows: a row satisfies (A, B, s) if B is shifted s places to the right cyclically from A. Thus a row with the form $(...., A, B, ...)$ satisfies $(A, B, 1)$, $(...., A, *, *, B, ...)$ satisfies $(A, B, 3)$ and so on. We know that for any $A, B \in [0, p-1]$ and $1 \leq s \leq p-1$ we have exactly one row of F_p satisfying (A, B, s).

Given a set of permutations Γ on an alphabet $\mathcal{A}$ we refer to the number of rows of Γ that satisfy (A, B, s) as the multiplicity of (A, B, s) in Γ. When Γ has the property that the multiplicity is independent of A, B and s we will refer to the (constant) number of rows satisfying (A, B, s) (for any A, B and s) as the *multiplicity of* Γ.

Theorem A venetian rectangle V_{p+k} constructed as above has $(p+k-1)!/(p-2)!$ rows and multiplicity $(p+k-2)!/(p-2)!$. More precisely:

$\forall A, B \in \mathcal{A}, \forall s \in Z_n$ the number of rows of V_{p+k} satisfying (A, B, s) equals $(p+k-2)!/(p-2)!$

Proof. By induction: we assume that the property holds for V_{p+k}, and proceed to show that it holds for V_{p+k+1}.

The base case: we need to show that the result holds for $k=0$ i.e. for F_p, but this is immediate from the property of the florentine square.

The induction step: The venetian rectangle V_{p+k+1} clearly has $p+k-1$ times the rows of V_{p+k}, which, by induction, is $(p+(k+1)-2)!/(p-2)!$. We now need to show that $\forall A, B \in [0, p+k]$ and $1 \leq s \leq p+k$, V_{p+k+1} has $(p+(k+1)-2)!/(p-2)!$ rows such that (A, B, s).

Consider first the case: $A, B \in [0, p+k-1]$, *i.e.* where both A and B are elements of V_{p+k}. For each row of V_{p+k} satisfying (A, B, s), we can add the "$p+k$" in $p+k-s$ places, *i.e.* in all the places not between A and B, to produce a row of V_{p+k+1} satisfying (A, B, s). For the rows of V_{p+k} satisfying $(A, B, s-1)$, there are $s-1$ places that we can insert "$p+k$" between A and B to produce a row of V_{p+k+1} with (A, B, s). Thus V_{p+k+1} has $p+k-1$, (*i.e* $p+k-s+(s-1)$) times as many rows as V_{p+k} satisfying (A, B, s). By induction, the total is $(p+(k+1)-2)!/(p-2)!$.

Note that if $s=1$ we ignore the second clause: we cannot get any $s=1$ rows by inserting entries into an $s=0$ row. If $s=p+k$ then we ignore the first clause: we take the row of V_{p+k} for which $(A, B, p+k-1)$ holds. There are $p+k-1$ places we can insert $p+k$ between A and B to give $p+k-1$ rows with $s=p+k$.

Now consider the case that $A = p+k$ and $B \in [0, p+k-1]$, (note we always have $A \neq B$ and since the role of A and B is entirely symmetric we can just consider $A = p+k$ without loss of generality):

For any given $1 \leq s \leq p+k$ we see that each row of the V_{p+k} will give rise to exactly one row of V_{p+k+1} for which (A, B, s) holds, essentially because each row of V_{p+k} is a (cyclic) permutation of $[0, p+k-1]$. By induction V_{p+k} has $(p+k-1)!/(p-2)!$ rows so again we get a multiplicity of $(p+(k+1)-2)!/(p-2)!$.

This completes the induction.

□

Of course, when we reach to next prime, q say, we can replace V_q by F_q.

The above construction gives us venetian rectangles with the balance property: $\forall A, B \in [0, n-1], s \in [1, n-1]$, the number of rows satisfying (A, B, s) is a constant. It does not however give us a flat distribution for the probability of any symbol falling in any particular row. For this, as previously for the florentine squares, we need to add a final random cyclic substitution to each row.

We could go ahead and use the above construction to form ballots with three onions, Θ_k, Θ_s and Θ_m, where the Θ_m encodes the positions of the P terms. Receipts would then have the form:

$$(i, \Theta_k, \Theta_s, \Theta_m)$$

We could put such terms through a re-encryption mix, but this would have the drawback that the indices would be left invariant by the mix. However, it is quite straightforward to enhance the construction to eliminate this problem. To this end we introduce an additional cyclic shift by position: after the row has been constructed from the P sequence, the appropriate row of F_p, and the values substituted by the modular addition by s_2, we rotate the row s_1 steps to the left cyclically, that is, we move every element up s_1 places on the ballot, cycling around to the bottom as necessary. The complete algorithm is shown in Protocol 2.

Protocol 2 Venetian rectangle construction - pairwise symmetry for composites

```
1: Choose a random list $P$ of $m$ distinct elements of $[0, n-2]$. Call them $c_p, c_{p+1}, ..., c_{n-1}$. Place elements $\{p, \dots, n-1\}$ in positions $c_p, c_{p+1}, ..., c_{n-1}$ respectively.
2: Choose $k \in_R [1, p-1]$. Place the elements of the $k$th row of $F_p$ in order in the remaining positions on the ballot.
3: Choose $s_2 \in_R [0, n-1]$. Add $s_2$ to each element, mod $n$.
4: Choose $s_1 \in_R \mathbb{Z}_q^*$. Rotate the row by $s_1$ to the left.
```

Lemma 2. *The venetian rectangle construction (Protocol 2) is a symmetrical ballot construction.*

Proof. That each element's position is uniformly distributed follows immediately from the uniformly chosen modular addition in Step 3.

Now consider condition 2. We need to prove that, for all pairs of candidates A and B, B's location is uniformly distributed given A's, over locations other than A's.

Firstly we observe that the balance property of V_n resulting from steps 1 and 2 of the protocol asserts that the multiplicity is independent of the A and B chosen. consequently we can treat the As and Bs as dummy variables and freely substitute them with a 1-1 mapping leaving the balance property invariant. It follows that the permutation on the candidate indices we apply in step 3 preserves the balance property.

We index the rows of V_n by the tuple $(k, s_1, s_2, c_p, \ldots, c_{n-1})$, with the cs pairwise distinct. We now show that there is a 1-1 correspondence between the rows of V_n and the elements of the tuple space:

$$\begin{aligned} S := \quad & \{k, s_1, s_2, c_p, \ldots, c_{n-1} \mid i, j \in [p, n-1], \\ & i \neq j \Rightarrow c_i \neq c_j\} \end{aligned}$$

This follows straightforwardly using an argument much like that presented in section 3.

Now, as long as we chose randomly with a flat distribution from S we guarantee the following:

$$\begin{aligned} & \forall i, \forall A, B \in \mathcal{A}, \forall s \in Z_n, A \neq B, \\ & \qquad \mathit{Prob}(V_{x+s \pmod n} = B | V_x = A) = 1/(n-1) \end{aligned}$$

Where $\mathcal{A}$ denotes the alphabet of symbols used to index the candidates, V_x denotes the value occupying position x and $\mathit{Prob}(X|Y)$ denoted the conditional probability of X given Y.

This guarantees again that the attacker, by shifting the position of the Xs, cannot do better than randomise the resulting values.

$\square$

4.2 Tabulation

We now describe how the votes should be tabulated to avoid leaking too much information.

Now ballots carry $m + 3$ onions:

$$(\Theta_k, \Theta_{s_2}, \Theta_{s_1}, \Theta_p, \Theta_{p+1}, \ldots, \Theta_{n-1})$$

That is, we encrypt each of the c positions separately. Receipts have the form:

$$(i, \Theta_k, \Theta_{s_2}, \Theta_{s_1}, \Theta_p, \Theta_{p+1}, \ldots, \Theta_{n-1})$$

Tabulation on the Bulletin Board proceeds as follows: first we absorb the index value into the s_1 onion in the usual fashion:

$$\Theta_i := \Theta_{s_1} \cdot \{i\}_{PK}$$

to give tuples of the form:

$$(\Theta_k, \Theta_{s_2}, \Theta_i, \Theta_p, \Theta_{p+1}, \ldots, \Theta_{n-1})$$

The Θ_i term is an encryption of the index value corrected for the rotation by s_1. Call this corrected index value $\hat{\imath}$. The resulting $m+3$ tuples of onions are put through re-encryption mixes. After the mixes, we perform a *Plaintext Equivalence Test* (PET) test of the Θ_is against each of the $\Theta_p, \Theta_{p+1}, \ldots, \Theta_{n-1})$. Where we find a match, between Θ_i and Θ_j say, this means that the index term that has been corrected for the s_1 rotation, corresponds to one of the P positions. If the match is with Θ_j, this means that the voter chose the slot into which the jth candidate was inserted, prior to the cyclic substitution. We can now correct for the substitution by forming $\widehat{\Theta} := \{j\}_{PK} \cdot \Theta_{s_2}$ as before.

Where we do not find a match, we need, as before, to determine how many of the cs are less that $\hat{\imath}$ in order to determine the correct column of the florentine square. This can be accomplished by forming encryptions of $\hat{\imath}-1, \hat{\imath}-2, \ldots, \hat{\imath}-n+1$ by computing:

$$\Theta_i \cdot \{-1\}_{PK}, \ldots, \Theta_i \cdot \{-n+1\}_{PK}$$

We now shuffle these and run (PET) tests of these against the $\Theta_{c_p}, \Theta_{c_{p+1}}, \ldots, \Theta_{c_{n-1}}$. The number of matches tells us how many of the c values are less than $\hat{\imath}$ and we now correct $\hat{\imath}$ accordingly. Call this corrected index value g. We can compute the corresponding $\widehat{\Theta} = (\Theta_k)^g \cdot \Theta_{s_2}$ as before. Finally we put all the $\widehat{\Theta}$ terms through a final mix and decrypt to reveal the original votes.

4.3 Receipt Freeness

The same simple coercion resistance strategy as in Section 3.4 works for this construction too. The voter simply rotates the list of candidates on the ballot until their checkmark is consistent with what the coercer demanded, then claims that that was the ballot permutation.

Consider what is revealed about each anonymised ballot:

1. If the corrected index $\hat{i}$ matches one of $c_p, c_{p+1}, \ldots, c_{n-1}$, then this fact is revealed (without showing which one matched).

2. Otherwise, we reveal how many of the $c_p, c_{p+1}, \ldots, c_{n-1}$ occurred before $\hat{i}$.

This implies that

1. all votes for anything in $\{c_p, c_{p+1}, \ldots, c_{n-1}\}$ (before the cyclic shift by s_2) are indistinguishable, and

2. for all $C \leq m$, all votes that place a checkmark after C inserted values are indistinguishable.

The coerced voter's receipt, combined with the permutation he tells the coercer that he saw, might allow her to infer exactly all of the ballot parameters. However, she is still overwhelmingly likely to find on the bulletin board a(nother) vote consistent with the story he told her, even if he told her a rotation of the true ballot permutation. If he claimed to have voted for one of $\{c_p, c_{p+1}, \ldots, c_{n-1}\}$ (before the cyclic shift by s_2), then the coercer will be satisfied as long as at least one other voter did so; if he claimed to have placed the checkmark at a point after C inserted items, then all he needs to assume is that some other voter actually did so. (And in many cases the value of C cannot be determined from a particular ballot permutation anyway, because there are multiple values of s_1 that produce the same result.)

In summary, the coercion resistance strategy is likely to be successful whenever the number of votes is much more than n. It seems impossible to do better than this and also reveal candidates' tallies—if there are candidates that absolutely nobody votes for, then a coercer can always pay a voter to choose one of them. In practice this threat is small enough to be ignored for almost all reasonable elections.

5 Discussion

5.1 Stronger Attack Models

So far we have been considering an attacker who tries to manipulate the outcome by altering the position of the X on receipts, or equivalently, altering the index values. The construction presented above ensures, as long as the attacker has no access to the k values on the receipts, he cannot do better than a randomisation attack. Thus the attacker can at best shift the outcome towards a draw. Now we consider attackers who attempt other manipulation of receipts and cryptographic terms involved in the tabulation.

An attacker with access to receipts may try to alter the k and s onions. Our constructions are not so effective against such an attacker. Examination of F_p for example shows that an attacker who can shift the k and s values can do better than random. If he wants to shift s by δ_s and k by δ_k for example he can simply multiply Θ_s by $\{\delta_s\}_{PK}$ and Θ_k by $\{\delta_k\}_{PK}$. In effect this is a move δ_s to the right and δ_k down on a the florentine square. Of course, the attacker doesn't know what square he starts from, but the balance property does not hold with respect to an attacker able to execute such moves. Nor is it clear how one might fix the florentine square construction to reestablish a balance property with respect to such moves. However, an attacker who could perform such manipulations could presumably just overwrite the onions with some other, more politically desirable, encrypted values anyway.

We can counter such an attacker by making it hard to manipulate the onions, which we can do by ensuring that all onions are signed prior to the start of voting. Such ballot signatures can be pre-audited before the election. This is in contrast to the signatures applied to the receipts once the voter has made her mark. Clearly, receipt signatures cannot be constructed until after the voter has made her choice and, consequently, we cannot pre-audit such signatures. This means that the receipt signatures are harder to validate that the onion signatures. There is the danger that the device that signs the receipts may be corrupt or faulty. This would result in valid complaints by voters being regarded as malicious. Consequently, it is necessary to introduce procedures to validate receipt signatures immediately after casting in the polling station so that a corrupt signing device is quickly identified and removed. This complicates the voting procedure for the voters and the officials.

The upshot of this is that it is much easier to ensure the integrity of the onions than that of the index value. Our florentine squares construction does not counter attempts to shift votes by manipulating the onions, but then the argument above suggests that it is easier to detect such manipulation.

We need also to consider attempts to manipulate the *fused* onions, that is the $\widehat{\Theta}$ onions computed from the receipts. Again, our construction does not counter the possibility of shifting votes by adding a constant shift to the fused onions. Here again though we can argue that the audit mechanisms applied to the Bulletin Board will with high probability detect such manipulation. Firstly note that the $\widehat{\Theta}$ terms are computed in a universally verifiable fashion from the receipts. From then on we employ the usual techniques to detect any manipulation as these terms go through the re-encryption mixes. Partial Random Checking [JJR02] only gives a 50% chance of detecting any single manipulation but we can improve this by using for example the Neff style verification of mixes, [Nef01], or indeed by using several mixes in parallel and applying RPC to each independently.

In summary, the integrity of position of the Xs seems to be the hardest element of the ballot processing to protect. The construction we propose here serves to blunt the effectiveness of any such attack by ensuring that an attacker can at best randomise votes by such manipulation.

5.2 Defining Recipt Freeness

An earlier version of this work contained a venetian rectangle construction similar to that in Section 4.1, but with no row rotation. If a coercer saw a voter's receipt and demanded to know what permutation had been printed on the ballot, there was a coercion resistance strategy: the voter had to compute the difference d between the candidate (number) the coercer demanded and the candidate he actually voted for, then add $d \bmod n$ to each other candidate on the ballot, and tell the coercer that that was the candidate order.

Should this variant be regarded as "receipt free" or not? We suggest that it should not, because in practice a coercer could succeed in this attack knowing that the computation was infeasible for an ordinary voter.[1] Of course, this also assumes some competence on the coercer's part—we assume she can detect when the voter claims an invalid ballot permutation. This seems more reasonable because she could consider the question long after the vote, while the voter may have to generate the lie on the spot.

Our current construction requires the voter to perform a much simpler coercion resistance strategy, simply rotating the list of candidate names by the appropriate amount. Nevertheless, we acknowledge that for some voters, in some elections, this may be too difficult. The exact definition of a human-computable coercion resistance strategy we leave as an interesting open problem.

[1]We thank an anonymous reviewer for this insight.

6 Conclusions

This paper addresses a vulnerability of the Prêt à Voter schemes that use simple cyclic shifts: an adversary able to manipulate index values on receipts before they are entered into the tabulating mixes, can shift a skewed distribution from one candidate to another. Firstly we have shown that the simple fix of introducing full permutations introduces coercion threats. Instead we have introduced a construction based on the notion of florentine squares that provides a combinatorial balance property that ensures that an adversary can now, at best, randomise ballots for which he is able to manipulate the index. The algebra of the florentine squares meshes nicely with the homomorphism of Paillier or exponential ElGamal allowing us to perform the tabulation without revealing anything about the set of permutations that appears on the voted ballots.

The florentine square construction only works for prime numbers. Given that we probably can't legislate for all elections to be conducted with a prime number of candidates, and it may not be allowed to add a number of null candidates, we have presented an extension to the florentine squares to deal with composite numbers of candidates. This is not as elegant as the construction for prime numbers of candidates. The decryption process is less efficient, but still fully hiding.

Our explorations here have also thrown light on the subtleties of the notion of coercion-resistance. It is not enough just for a coercion-resistance strategy to exist, it must also be extremely easy for voters to understand and execute. We have observed that Prêt à Voter is potential vulnerable to Italian style attacks in which the voter is required to record the candidate order on the ballot. The best situation is where no information is revealed about the actual, ballot permutations during tabulation and all permutations are potentially valid. In this case the voter always has the very simple strategy of switching his vote and the coercer's. If some information about the set of actual permutations is revealed during tabulation then there may the possibility that this strategy is not guaranteed to work. Also, if the set of valid permutations is constrained, then the voter's strategy must be such that it will always yield a valid permutation. In this paper we have ensured that rotation of the candidate list will always yield a valid permutation. The simple flipping of two candidates will only work if the full set of permutations is allowed.

7 Acknowledgements

Thanks to Alina Vdovina, Ian Wanless and Kim Ramchen.

References

[CD07] Charles Colbourn and Jeffery Dinitz, editors. *Handbook of Combinatorial Designs.* Chapman and Hall (CRC), 2007.

[CRS05] D. Chaum, P. Y. A. Ryan, and S. A. Schneider. A practical voter-verifiable election scheme. In *Proc. European Symposium on Research in Computer Security (ESORICS)*, pages 118–139. Springer, 2005. LNCS 3679.

[DKR05] S. Delaune, S. Kremer, and M. Ryan. Receipt-freeness: formal definition and fault attacks. Workshop on Frontiers of Electronic Elections, 2005.

[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.

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

[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.

[PR06] T. Peacock and P. Y. A. Ryan. Coercion-resistance as opacity in voting systems. Technical Report CS-TR 959, Newcastle University, 2006.

[RS06] P.Y.A. Ryan and S. Schneider. Prêt à Voter with Re-encryption Mixes. Number 4189. Springer-Verlag, 2006.

[Rya04] P. Y. A. Ryan. A variant of the chaum voter-verifiable scheme. Technical Report CS-TR-864, School of computing science, University of Newcastle, 2004. www.cs.ncl.ac.uk/research/pubs/trs/papers/864.pdf.

[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.

[XSHT08] Zhe Xia, Steve A. Schneider, James Heather, and Jacques Traor'e. Analysis, improvement, and simplification of Prêt à Voter with paillier encryption. In *Proc. USENIX ACCURATE Electronic Voting Technology Workshop*, 2008.