Implementing STV securely in Prêt à Voter

James Heather

Department of Computing, University of Surrey

E-mail: j.heather@surrey.ac.uk

Abstract

Work on electronic voting systems to date has largely focused around first-past-the-post voting. However, the governments of many countries, and many non-governmental organisations, use a single transferable vote system, in which the voter needs to indicate not just a single preferred candidate but a preference ranking of (some or all of) the candidates on offer.

This paper investigates the possibility of modifying Prêt à Voter to cope with a single transferable vote system. With its newer form involving re-encryption mixes [8], this seems at first sight to be impossible; with the older version based on RSA onions [2], the obvious approach works, but security is less than ideal; using multiple re-encryption mix onions for each vote, combined with a lazy decryption semantics, however, proves to be an elegant and efficient solution to the problem.

1. Introduction

Over the last few years, several electronic voting systems have been introduced that appear to offer a high level of security. We have voter-verifiable receipts that yet cannot be used to prove to a third party how one voted; we have decryption and tallying processes that are publicly verifiable and yet do not allow individual encrypted votes to be matched with their decrypted counterparts; we have distributed decryption phases that do not allow a single corrupt machine to rig the election.

What is rather harder to find in the literature, however, is the flexibility to cope with different voting and tallying systems. Most of the energy expended to date has been focused on *first-past-the-post* (FPTP) systems, in which the voter simply chooses a single candidate for election. Encoding this information is typically a small enough task that some attractive properties can be maintained: Prêt à Voter allows re-encryption mixes that keep the onion size from being affected by the number of tellers, and also allows the

random partial audit of the teller decryption phase to be run arbitrarily often and with different links being uncovered each time; Punchscan [5, 1] allows a mark daubed through a small hole to encode the information cryptographically on each of two sheets but so that both sheets are needed to read the vote without decrypting; ThreeBallot [6] allows the vote to be encoded by having the selected candidate's name appearing on more sheets than the other names.

A notable exception here is Clarkson and Myers' work [3] on modifying the re-encryption mix incarnation of Prêt à Voter to deal with Condorcet elections. We shall have more to say about this in Section 4.1.

The preponderance of FPTP e-voting systems is perhaps explained by the fact that the origin of many of these systems is to be found in the UK or the USA, where both national and local elections are run by FPTP. However, a glance at the international scene shows that FPTP is far from being the only voting system in real use. The *single transferable vote* (STV) is used in the Republic of Ireland for all national elections, and also for local and European elections; STV is also the method of electing Senate and for certain State elections in Australia; India uses it to elect most members of its Upper House; Malta uses STV for all elections; in the UK, although all English elections and Parliamentary elections are run by FPTP, STV is employed in Northern Ireland and Scotland for local elections; and even in the USA, which is heavily dominated by FPTP, some official elections in Cambridge, MA, and Minneapolis, MN, are run by STV. When one considers non-governmental organisations, STV appears even more popular: it is used by student unions throughout the UK and increasingly in the USA; for election to the General Synod of the Church of England; and in countless other organisations. It is clear that any e-voting system that wants a large potential market needs to be able to deal effectively with STV.

This paper constitutes a detailed discussion on how to modify Prêt à Voter to cope with STV. Accordingly, Section 2 contains a brief summary of how STV works; in Section 3 we then give an overview of the Prêt à Voter system, with the original RSA onions and with the newer El-

☐	Liesl
1	Martha
3	Louisa
☐	Gretl
2	Brigitta

Figure 1. An STV ballot form

Gamal re-encryption mixes; the main contribution of this paper then comes in Section 4 where we highlight the problems with a simplistic attempt to incorporate STV into Prêt à Voter and then show how to overcome them; we then give conclusions in Section 5.

2. The STV system

The basic notion to STV is, as the name suggests, that one has only a single vote, but it may be transferred from one candidate to another. This transfer is done when one's favoured candidate is eliminated from the race; so one marks (with a '1' rather than with a '×') one's chosen candidate—say, Martha—but also indicates (with a '2') which candidate one would like if Martha were to be eliminated from the contest, and then one marks one's next best option (with a '3'), and so on (see Figure 1).

Tallying is done by initially sorting the piles of ballot papers according to first preferences. Each pile is counted, and the candidate with the fewest first preferences is eliminated from the contest. The ballot papers in that candidate's pile are then redistributed among the other piles according to the second preferences indicated on the papers. (In the example in Figure 1, if Martha were eliminated, this vote would then go onto Brigitta's pile.)

The piles are then counted again (with a transferred vote given the same weight as a first-preference vote), and the candidate with the smallest pile is once more eliminated, and the votes are transferred again.

Three points of clarification are needed:

1. if a ballot form does not specify a ranking all the way down, and all its specified candidates have been eliminated, the ballot paper is then discarded;
2. if two or more candidates are in equal last place, the candidate to be eliminated may be chosen by drawing of lots, or by some other method (for example, counting second preferences) according to the rules of the specific election;

Louisa	
Gretl	
Brigitta	
Liesl	
Martha	
	3eQQv57

Figure 2. A Prêt à Voter ballot form

3. usually an STV count includes the notion of a *quota*, which essentially determines the number of votes needed such that one cannot be beaten regardless of how the rest of the count goes. If there is only one candidate being elected, then the quota will be just over half the votes, but it will be fewer if there is more than one seat to be filled. When a candidate reaches the quota, that candidate is declared elected, and his surplus votes (the votes he has in excess of the quota) are transferred, according to any of a variety of mechanisms.

Discussion about STV elections for multiple seats with fractional vote transfers is deferred to Section 5. For the main body of the paper, we shall assume that votes maintain their full value whenever they are transferred. This is always the case in STV when a single winner is to be returned.

The key points to note for this paper are that one's vote consists of a (possibly partial) ranking rather than a single choice; that it is not enough simply to record how many *n*th preferences each candidate has attracted for each *n*, because when transferring votes we need to identify the second preferences indicated on the specific ballot papers being discarded; and that in many cases some of the preferences indicated on a ballot paper might not be used in the count because the tallying process might finish (that is, a winner might be declared) without the vote being transferred all the way down to the last preference.

3. Prêt à Voter

The purpose of this section is to remind the reader of the overall structure of Prêt à Voter and to recap on the salient points. Space considerations forbid rehearsing every detail here, and some familiarity with the system is assumed; for a complete description the reader is advised to consult [2, 8].

A Prêt à Voter ballot form (suitable for FPTP) is shown in Figure 2. The candidate list is given on the left-hand side, and boxes are provided on the right-hand side for the voter to cast his vote. When he has finished, he tears the

ballot form down the middle and shreds the left-hand portion containing the candidate names; the right-hand portion, containing the vote, is fed into a scanner and the vote is recorded; this right-hand portion is then retained and forms the receipt.

The candidates are listed in some canonical ordering, but with a random cyclic shift applied, with different ballot forms having different shifts; consequently, the scanner, which sees only the right-hand side, cannot interpret the vote, and nor can the voter prove to a third party how he voted. The cyclic shift is, however, embedded in the cryptogram at the bottom of the right-hand side of the ballot form, so that the vote can be recovered later for the tallying process. This cryptogram, the *onion*, has been encrypted either several times, once each with the public keys of several different machines (the *tellers*), or with a threshold key whose decryption key is shared by the tellers, so that all of these machines must participate in the decryption process. The votes are processed all at the same time (or, at least, in large batches), and when a teller processes a batch of forms, it mixes them before passing them on to the next teller. This ensures that when the votes have all been decrypted it is impossible for anyone to match an encrypted vote with a decrypted vote. A mechanism is provided so that this decryption process can be audited, and it can be verified (neglecting a vanishingly small probability) that the set of decrypted votes matches up (as a whole) with the set of encrypted votes.

The precise nature of the encryption differs between the two incarnations of Prêt à Voter. In each case, what follows describes how Prêt à Voter works in the case of a FPTP election, where a voter is required simply to choose a single candidate.

3.1. Prêt à Voter with RSA onions

If there are n tellers, then the onion contains $2n$ layers. Construction of a ballot form means construction of a pair (r, x) , in which r is the shift that has been applied to the candidate list on the form, and x is the multi-layered onion. We start with $r = 0$ and $x = \langle \rangle$, and apply the following procedure n times, once for each teller.

Suppose that the ballot form is currently (r, x) . We generate a new random cyclic shift g , which will be an integer in the range $0 \leq g < k$, where k is the number of candidates. This value is called a *germ*. The new ballot form becomes

$$(r \oplus_k g, \{g, x\}_{PK_i})$$

in which ' $\oplus_k$ ' represents addition modulo k , and ' PK_i ' denotes the current teller's public key, and the encryption is assumed to contain enough redundancy to foil guessing attacks. A second layer for the same teller is then added; the

shift/onion pair now becomes

$$((r \oplus_k g) \oplus_k g', \{g', \{g, x\}_{PK_i}\}_{PK_i})$$

and we now move onto the next teller¹. When all is done, the first component gives the cyclic shift to be applied to the candidate ordering on the printed form, and after printing this value is discarded; this value can be recovered by the teller chain as they iteratively strip off layers of the onion and sum the embedded g -values (modulo k).

This results in quite large onions if there are several tellers and the various security parameters (the amount of redundancy included in the encryption) are high enough. Accordingly, what is printed on the bottom of the right-hand half of the ballot form is a signed hash of the onion. The mapping from hashes to onions can be made public on a web bulletin board.

The decryption phase involves sending the first teller the value v of the vote, which will be an index into the shifted candidate list as it appears on the ballot form, and the onion. The teller removes the first layer, recovers the germ, and subtracts this from v (modulo k). This gives a new vote value, v' , which represents the same vote but according to the cyclic shift determined by the germs in the remaining layers of the onion. The teller then applies the same process to the second layer, and passes the resulting v'' and smaller onion onto the second teller. What is left at the end of the teller chain is a vote value representing the same vote but in the canonical ordering.

Votes are decrypted in large batches. The teller decrypts the outer layer of each vote, randomly reorders the batch, and sends the result back to the web bulletin board; it then removes its other layer, shuffles the batch again, and sends the result back once more.

The purpose of giving two layers to each teller is to allow random partial auditing of the teller's actions, by forcing the teller to reveal a random half of the first shuffle, and the remaining half of the second shuffle, and prove that these shuffles are as claimed. This gives no way of matching an encrypted vote through the teller chain to an unencrypted vote, but reduces the probability of the teller getting away with manipulating n votes to 2^{-n} .

3.2. Prêt à Voter with re-encryption mixes

The newer version of Prêt à Voter uses ElGamal to store the encrypted cyclic shift. The mathematics of ElGamal encryption is unimportant for this paper, but three key features need noting.

¹The original description of Prêt à Voter has two separate public keys for each teller, one for the outer encryption layer and one for the inner layer. This does not appear to add any extra security, however, and so here we stick to one key per teller.

1. An ElGamal encryption of a message m contains buried within it a random value z of the same length as the plaintext. The encryption, $\langle m, z \rangle_{PK}$, is twice as long as m .
2. Given an ElGamal encryption $\langle m, z \rangle_{PK}$, it is possible to change this value of z even without knowing m or z . Thus, given just the ciphertext and the public key, one can produce a ciphertext that looks different to anyone not having the corresponding secret key, but decrypts to the same value. This is known as *re-encryption*.
3. Given an ElGamal encryption $\langle m, z \rangle_{PK}$, it is also possible to multiply m by a constant, within the encryption, also without knowing anything except the ciphertext and PK . In other words, one can produce $\langle p \cdot m, z \rangle_{PK}$ from $\langle m, z \rangle_{PK}$ without knowing m or z . In Prêt à Voter, the secret value s is always encrypted as $m = \gamma^s$ for some fixed and public γ ; the effect of this is that now one can add a constant t to s inside the encryption by using the aforementioned technique to multiply the plaintext by γ^t .

From this point onwards, we will write ' $(s, z)_{PK}$ ' as shorthand for ' $(\gamma^s, z)_{PK}$ '. The two points given above mean that from $(s, z)_{PK}$ and PK it will be possible to produce another value $(s + t, z')_{PK}$ that looks unrelated to anyone not possessing the decryption key.

The ElGamal version of Prêt à Voter uses ballot forms containing an 'onion' with a single layer; if the cyclic shift applied to the candidate list is g then the onion will be $(-g, z)_{PK}$ for some z . (The onion is produced by a distributed process so that no single entity knows the value of g , but this need not concern us here.) The decryption key is a threshold key, and each teller has a share of it; an appropriately large numbers of tellers, therefore, need to act together to decrypt an onion.

The anonymising/mixing phase and the decryption phase are kept separate in this version of Prêt à Voter. Before the mixing phase takes place, each ballot form, containing a vote v and an onion $(-g, z)_{PK}$, is changed to contain a single value of $(v - g, z)_{PK}$. Now $v - g$, taken modulo k , represents the vote in the canonical ordering.

For the anonymising/mixing phase, the first teller receives a batch of ballot forms, each now containing a single onion into which the vote value has been absorbed. The teller re-encrypts each onion, randomly reorders them, and sends them back; it then performs the same process, re-encrypting and reordering them once more, and sends them back again. The batch is then passed to the next teller. When it has passed right through the teller chain, the mixing process is complete. A random partial audit can be performed in exactly the same way as with RSA onions, with one neat twist: in the case of a dispute, the anonymising mix can be

People's Front of Judea 1	
Judean People's Front 1	
People's Front of Judea 2	
Judean People's Front 2	
Judean People's Front 3	
People's Front of Judea 3	

Figure 3. An STV ballot form in the canonical ordering

	1
	6
	3
	2
	4
	5

Figure 4. A cyclic shift leaks too much

performed as many times as is required, with a new audit each time, because with each run the intermediate values will be different.

The decryption phase simply involves the tellers using their shares to decrypt each (anonymised) vote. This gives a complete set of decrypted votes in the canonical ordering, and the votes can then be tallied in the usual way.

4. STV and Prêt à Voter

At first sight, the obvious attempt to modify Prêt à Voter to deal with STV involves keeping the cyclic shift of the candidate order to protect anonymity, but changing the information embedded in the vote itself to something that represents a (possibly partial) ranking of candidates.

However, this leaks too much information. (This is suggested in [2] but not demonstrated.) Suppose the ballot form, in the canonical ordering, reads as in Figure 3. If a random cyclic shift is applied to this ballot form, and we afterwards see an encrypted receipt that looks like Figure 4, it is not difficult to make a good guess as to what the cyclic shift was. There is a strong likelihood that this vote is going to rank the three candidates representing the People's Front of Judea above their bitter rivals from the Judean People's Front, or vice versa; the only cyclic shift that fits with this is the one with 'Judean People's Front 1' as the name at the top of the ballot form. What is needed is for the ballot form (and, hence, the onion) to contain a random *permutation* of

candidate names.

4.1. Re-encryption mixes

Work by Clarkson and Myers [3] has already shown how Condorcet can be incorporated into Prêt à Voter in its re-encryption mix incarnation. In a Condorcet election, each voter submits a complete ranking of the candidates according to preference; tallying is then done by making pairwise comparisons between candidates and determining, for each pair, which would win in a straight vote between the two. Each ballot form is considered, and the number of ballot forms with candidate A ranked above candidate B is compared with the number ranking B above A. When all pairs have been considered, the candidate who wins most pairings is declared the winner.

Clarkson and Myers solve the problem by treating a ranking of k candidates as being $\frac{1}{2}k(k-1)$ miniature ballot forms each expressing a preference between two candidates. Although this generates $O(k^2)$ onions, each onion is small because the ElGamal ‘onions’ are single layer and their size does not vary with the number of tellers involved. The approach is attractive because it eliminates the possibility of looking for the occurrence of a particular ballot form when the decryption phase is finished: there is nothing to connect one decrypted pairwise ranking to another that originated on the same ballot form.

This is important because it avoids the possibility of using lower-ranked (and, thus, less important) preferences as a covert channel in a system like Prêt à Voter where the decrypted ballot forms are made publicly viewable after the election. We shall return to this point in Section 4.2.2.

Unfortunately, it does not appear to be possible to get a full permutation into an ElGamal onion. In order to preserve anonymity, an essential first step before the mixing is to fold the vote into the onion so that what is left is a vote according to the canonical ordering. This cannot be done here, because we would need to be able to take a completed encrypted ballot form constructed as $\langle \pi_g, z \rangle_{PK}$ along with a vote π_v , where π_g and π_v are now permutations, and produce $\langle \pi_v \circ \pi_g, z \rangle_{PK}$, in which ‘ $\pi_v \circ \pi_g$ ’ denotes the composition of the two permutations. There does not seem to be any way of representing these permutations in a way that is malleable inside an ElGamal encryption, because permutations cannot be encoded as integers in such a way that the composition of the two permutations is cashed out as the product or the sum of the two integers. (For a start, composition of permutations does not commute.)

4.2. RSA onions

With RSA onions, the story is different. Here, each onion layer is removed by its teller, and combining the vote

with the germ is done after decryption. This allows for encoding the permutation according to any reasonably concise scheme. Each layer i of the onion contains some permutation π_i , and the candidate ordering on the ballot form is then the product of all of these permutations, applied to the canonical ordering. At each stage of the decryption process, when a layer is removed and a germ π_i is recovered from the onion, the teller applies π_i^{-1} to the vote to get it into the ordering represented by the rest of the onion. At the end of the process, the vote has been transformed into a vote under the canonical ordering and tallying can proceed as before.

(It is suggested in [2] that STV can be dealt with in this system by using a publicly known hash of the germs embedded in the onions to create a permutation. We see no reason not to embed the full permutation in the onion, so that is the approach outlined above, but the analysis that follows applies equally well to either method.)

This simple switch of an index value for a permutation, in both the vote and the onion, is sufficient to make the system work with STV. However, there are problems with the security.

4.2.1 Tracking sets across the teller chain

A vote under STV need not be a full permutation (in contrast to Condorcet, for example). The ballot paper in Figure 1 is a valid ballot under the STV semantics; once the three indicated candidates have been eliminated, the ballot form is discarded and its vote is lost. However, this means that it is evident from the receipt how many candidates have been selected. Since the encrypted receipts and the decrypted votes are published on the web bulletin board, and since in each case one can see how many entries there are on the ballot form, it is possible to partition both sets according to the number of entries, and match these sets (as complete sets) between the encrypted receipts and the decrypted votes.

This problem may perhaps be regarded as an intrinsic property of STV. If the receipt is to be verifiable by eye then either it must show how many entries were made or some extra information must be encoded and checked; any solution will naturally complicate the issue and make the voter’s job harder. We will discuss a possible solution to this shortly. Let us note in passing, however, that automatically and randomly allocating rankings to boxes that the voter has left unfilled is out of court from the start: this might affect the result of the election. When we do the tallying, we will no longer be following the STV semantics; whatever the pros and cons of such a voting system, it is certainly not equivalent to STV, and since the aim is to implement STV, we cannot follow this path.

This matching of sets might also be considered fairly harmless in a large election if we are confident of large sets

Figure 5. Auditing in Prêt à Voter

Figure 6. Attempting to track a vote

in each case. There is an additional wrinkle, however.

Recall that the decryption phase takes batches of votes down the teller chain, with the votes being shuffled at each stage, and partial random audits employed to verify that the tellers are operating according to their brief. Each teller performs two decryptions and shuffles, and the random auditing reveals half the links for the first shuffle, and the corresponding half for the second shuffle. In Figure 5, auditing of a single teller is shown operating on ten votes. The teller has disclosed information to enable the revealed links, shown as dashed lines, to be publicly verified.

An attempt to use the links to track a vote from end to end is very quickly defeated by the large number of possible paths. Figure 6 has a single vote highlighted on the left-hand side; this gives rise to a single vote in the middle column; but half the votes on the right-hand side are possible matches. If there were several tellers, this effect would be quickly multiplied across the chain. A formal analysis of this in [4] provides clear evidence that this shuffling process does its job effectively.

But, as we have noted, an encrypted vote still reveals the number of preferences indicated on the ballot form. This is true not just for the initial encrypted vote but for its partial decryptions at every stage along the teller chain. We can partition the sets of votes all the way through.

Figure 7. Tracking an individual vote

Figure 7 shows a teller audit in which one of the partition sets has been identified and highlighted in each of the three columns. The darker vote in the left-hand column clearly must match with the darker one in the middle column, because this link has been revealed; but we see immediately that there is also only one possibility for its location in the right-hand column, because it has to be one of the votes in the partition set and it must be one whose link with the middle column is not revealed.

Even though there are four votes in the set, we have managed to link one on the left with one on the right. This is because the partition set has not been split in half by the choice of audit links. It would be more efficient to make sure that the audit links are chosen to force this to happen; or, equivalently, to run each partition set one by one through the teller chain and audit process.

However, we can do better. We can include a dummy ‘stop’ candidate in the election, whose name need not appear on the ballot form but is always implicitly in last place in the candidate list, and whose appearance in the ranking is to be taken as an indication that this is the point at which the voter made no further choices. What the booth does is to add this candidate’s index to the ranking as the next ranking after those specified by the voter, and then fill in the remaining places arbitrarily. This way, the encrypted vote and the fully decrypted vote still show how many candidates were selected (because the highest number always indicates the ‘stop’ candidate), but during the decryption process the random permutations will move this value around at each stage, and no-one will be able to identify it. End-to-end partitioning is still possible, but not tracking across the chain.

4.2.2 Covert channels

This method of implementing STV gives us no protection against the issue raised above of using lower-ranked preferences as covert channels.

Suppose we have an election involving ten candidates, one of whom is Alice. This means that there are $9! =$

Alice	1
Esther	2
Duncan	3
Harriet	4
Jonathan	5
Bob	6
George	7
Charlie	8
Isobel	9
Fred	10

Figure 8. ‘The Italian Attack’

362,880 different complete rankings of candidates that put Alice at the top. She can coerce Norman to vote for her as his first preference by stipulating a particular one of these rankings (say, the ranking given in Figure 8) that she wants him to enter on the form, on pain of unpleasantness to follow if he disobeys. Although she cannot tell from his receipt whether he has done her bidding, she can check the decrypted votes afterwards to see whether this exact vote sequence appears anywhere, and take her frustration out on Norman if she cannot find it. Unless the electorate is huge, this exact ranking is unlikely to appear by chance; certainly Norman would be a brave man to take the risk.

This is a consequence of the attempt to split the problem into two stages: first decrypt the (anonymous) votes, and then tally them. Any system that publishes the full decrypted votes will be subject to this type of attack.

The attack given here is not new—it is sometimes referred to as ‘The Italian Attack’—but it does not seem to appear in the literature anywhere.

For FPTP, there is really no sensible middle ground when it comes to decrypting a vote: either one completely decrypts it, or one does not decrypt it at all. And since the (usual) FPTP semantics require the vote tally for all candidates to be published, rather than simply the ranking of the candidates or even just the name of the winning candidate, it is evident that all of the votes will need to be decrypted. On the other hand, there is no equivalent of the Italian voting attack with FPTP because the information encoded in a vote is so little, so the problem does not arise.

With STV, however, the amount of information buried in a vote is much greater, but not all of the information will necessarily be required to complete the tallying. The information that is present in the entire set of decrypted votes but not needed for tallying falls under three headings:

Unused preferences. Some of a voter’s preferences never get considered because the vote is allocated to the win-

ner’s pile before all of the preferences are used. In the case of a single seat election, any rankings lower than that of the eventual winner will never be needed; even rankings above may not be needed if the eventual winner hits quota soon enough.

Transfer history. When ballot paper’s second preference is eliminated, and we uncover the third preference and reallocate the vote, we do not need to know what the first preference was; in other words, the transfer history of this ballot paper is irrelevant. As long as we have the information to transfer the right votes to the right piles, we can still tally accurately.

Absolute rankings. When a candidate is eliminated and his votes are transferred, we obviously need to know how to distribute these votes across the other piles. The next preference down on the list needs to be uncovered so that we can work out who should next receive the vote. What we do not need to know, however, is the absolute ranking that the transferred ballot paper specified for the eliminated candidate or for the new candidate. It makes no difference whether this is a first preference that has been ruled out and we are moving to the second preference, or this is a sixth preference that has been ruled out and we are moving to the seventh.

By publishing the full decrypted votes, we open ourselves up to attacks that use covert channels. Clearly what is wanted is a lazy evaluation semantics for the encrypted votes that releases the information only when it is required for the tallying.

Achieving this with RSA onions, it turns out, is harder and less elegant than returning to ElGamal and re-encryption mixes.

4.3. Multi-valued re-encryption mixes

The idea behind the solution is to encode the vote as a sequence $\langle c_1, \dots, c_k \rangle$ of cryptograms, each ultimately decrypting to a candidate identifier (such as an index into the canonical ordering of candidate names). Initially all the heads (first components) of the votes, representing the first preferences, are decrypted, and the votes are sorted into piles. The smallest pile needs redistributing; we take each vote, detach the head (representing the eliminated candidate), and pad the tail so that the length of the sequence is maintained, and decrypt the new head. These modified votes are then placed into the appropriate piles and each pile is anonymised and mixed so that the transferred votes cannot be distinguished from the original votes. We now count the piles once more, and continue as before. As soon as we reach a point where one candidate has more than half of the votes, we stop and declare a winner.

$\llbracket \text{Alice} \rrbracket_{PK_R}$	$\llbracket \text{Alice} \rrbracket_{PK_T}$
$\llbracket \text{Bob} \rrbracket_{PK_R}$	$\llbracket \text{Bob} \rrbracket_{PK_T}$
$\llbracket \text{Charlie} \rrbracket_{PK_R}$	$\llbracket \text{Charlie} \rrbracket_{PK_T}$
$\llbracket \text{Duncan} \rrbracket_{PK_R}$	$\llbracket \text{Duncan} \rrbracket_{PK_T}$
$\llbracket \text{Esther} \rrbracket_{PK_R}$	$\llbracket \text{Esther} \rrbracket_{PK_T}$
	$\llbracket \text{STOP} \rrbracket_{PK_T}$

Figure 9. Initial creation of the ballot form

4.3.1 Constructing the onions and votes

More precisely, each ballot form (for an STV election to return one winner from k candidates) contains a signed hash; this hash is used as a look-up to retrieve $k + 1$ ElGamal ‘onions’. The first k are ElGamal terms of the form $\langle i, z \rangle_{PK_T}$, where i is an index into the canonical ordering, z is a random value (different for each onion), and PK_T is a public key corresponding to a threshold ElGamal secret key that the tellers will use for decryption. The first k onions will cover values from $i = 1$ to $i = k$, but in a randomised order, corresponding to the order of the candidate names printed on the left-hand side. The last onion is called the ‘stop’ onion and it contains the value $i = 0$; it performs a function analogous to that of the ‘stop’ candidate of Section 4.2.1. The construction of the ballot form can be made subject to the same random auditing that is performed in the FPTP version of Prêt à Voter.

4.3.2 Distributed creation of the ballot form

Creation of the form is similar to that of FPTP Prêt à Voter with re-encryption mixes. We assume that we have a public key PK_R corresponding to a threshold ElGamal decryption key whose shares are held by a set of registrars; these registrars will be involved in the on-demand printing of a ballot form. The tellers also share a threshold key whose public complement is PK_T . We further assume some set of l clerks who will randomise the ordering on the ballot form.

Initially the ballot form is created as depicted in Figure 9. We use $\llbracket \text{Alice} \rrbracket_{PK}$ to represent an ElGamal onion encrypting Alice’s index in the (public) list of candidates in the canonical order, using the public key PK . On each side we have a sequence of onions that, when decrypted, together give an ordering of the candidate list. At the bottom, on the right-hand side, we also have a ‘stop’ onion.

The ballot form now gets passed through the chain of clerks. Each one chooses a random permutation and applies it to both sides of the ballot form, but leaving the ‘stop’ onion where it is. The clerk now re-encrypts each component so that no-one will be able to tell what the ordering is without the relevant decryption key. The ballot form we are

$\llbracket \text{Charlie} \rrbracket_{PK_R}$	$\llbracket \text{Charlie} \rrbracket_{PK_T}$
$\llbracket \text{Bob} \rrbracket_{PK_R}$	$\llbracket \text{Bob} \rrbracket_{PK_T}$
$\llbracket \text{Esther} \rrbracket_{PK_R}$	$\llbracket \text{Esther} \rrbracket_{PK_T}$
$\llbracket \text{Alice} \rrbracket_{PK_R}$	$\llbracket \text{Alice} \rrbracket_{PK_T}$
$\llbracket \text{Duncan} \rrbracket_{PK_R}$	$\llbracket \text{Duncan} \rrbracket_{PK_T}$
	$\llbracket \text{STOP} \rrbracket_{PK_T}$

Figure 10. A reordered ballot form

Charlie	
Bob	3
Esther	1
Alice	
Duncan	2
STOP	
	w4JKz

Figure 11. A completed ballot form

left with is as in Figure 10: it is as before but with the order permuted. Only the registrars can read the candidate list on the left-hand side; only the tellers can read the onions on the right-hand side.

The forms are made subject to the same kind of random audit check as in the FPTP version of Prêt à Voter.

Each ballot form is then stored on the web bulletin board, along with a signed hash. The form can be printed by any threshold set of registrars.

4.3.3 Voting

The voter fills in the ballot form exactly as before. The receipt contains the ranking in the order entered on the form, plus a signed hash of the sequence of teller onions (including the ‘stop’ onion).

The booth simply sends to the web bulletin board the information that is on the receipt. It does not even need to know the teller onions; it just sends the permutation for publication, and the signed hash so that the web bulletin board knows which ballot form is being used.

The web bulletin board first publishes this (possibly partial) ranking, along with the signed hash; the candidate order is random so this does not allow anyone to read the vote. It also converts the vote into a form suitable for processing by the tellers by using the ranking entries as indices into the sequence of teller onions it has stored.

Suppose the voter has completed the ballot form, and it now looks like the one in Figure 11. The ‘STOP’ row does

5	$[[\text{Charlie}]]_{PK_T}$	$[[\text{Esther}]]_{PK_T}$
3	$[[\text{Bob}]]_{PK_T}$	$[[\text{Duncan}]]_{PK_T}$
1	$[[\text{Esther}]]_{PK_T}$	$[[\text{Bob}]]_{PK_T}$
6	$[[\text{Alice}]]_{PK_T}$	$[[\text{STOP}]]_{PK_T}$
2	$[[\text{Duncan}]]_{PK_T}$	$[[\text{Charlie}]]_{PK_T}$
4	$[[\text{STOP}]]_{PK_T}$	$[[\text{Alice}]]_{PK_T}$
w4JKz	w4JKz	w4JKz

Figure 12. Converting the vote

not appear on the printed form, but it is implicitly always in last place; because it does not appear on the form, it can never have a ranking written next to it. Right at the bottom of the form is written the signed hash of the onion sequence stored by the web bulletin board.

The first step is to pad the ranking so that it is a complete permutation. The first unused rank is assigned to the ‘stop’ position; any remaining blank boxes are then filled in increasing order from top to bottom. (The order in which the boxes are filled does not make any difference, but it is important that it is done by a deterministic mechanism so that there is no possibility of allowing the web bulletin board to use its choice of ordering as a covert channel to leak information; filling the boxes from top to bottom is the easiest way of making the process deterministic.) The result is the first column in Figure 12. Then the web bulletin board looks up the teller onion sequence by using the hash, and retrieves the middle column of Figure 12. The web bulletin board cannot decrypt the onions, but it does know that their order corresponds to the candidate order on the ballot form. (It does also know that the ‘stop’ onion is in the last place, because this is always where it comes.) Now it produces the converted vote by reordering the onions it has retrieved according to the ranking it has just padded. The result is the third column in Figure 12. This gets published as it is, with no re-encryption performed, so that the conversion can be publicly verified. The burden on the individual voter is simply to check that the receipt matches the partial ranking that has been published on the web bulletin board; anyone can retrieve the onion list, check that the hash matches that on the receipt, and verify the reordering.

4.3.4 The anonymising mix process

The anonymising stage of the process runs analogously to standard Prêt à Voter with re-encryption mixes, except that to re-encrypt a vote it is necessary to re-encrypt each of the $k + 1$ entries of the vote (including the ‘stop’ onion, which might now be anywhere in the vote sequence). The teller takes in a batch of ballot forms, re-encrypts each component

Esther
$[[\text{Duncan}]]_{PK_T}$
$[[\text{Bob}]]_{PK_T}$
$[[\text{STOP}]]_{PK_T}$
$[[\text{Charlie}]]_{PK_T}$
$[[\text{Alice}]]_{PK_T}$

Figure 13. Uncovering the top preference

of each vote, and then mixes the batch (but leaves the intra-vote ordering intact). The anonymised and mixed batch is returned to the web bulletin board, and the teller repeats the process. The second batch is then returned, and is passed on to the next teller in the chain. This anonymising mix process is audited in precisely the same way as with FPTP and, as with FPTP, the whole anonymising mix and tallying process can be audited as many times as required.

4.3.5 Tallying

Decryption of the votes for tallying purposes is done on a strictly need-to-know basis. First, the hashes are removed, and all first preferences are decrypted (by threshold decryption, as in standard Prêt à Voter with re-encryption mixes), and the votes are sorted according to first preference. The encrypted vote in the right-hand column of Figure 12 thus becomes as in Figure 13: Esther’s name is now visible at the top. (The shaded elements of the sequence are encrypted onions, and so no-one, including individual tellers, can see inside them.)

The candidate attracting the fewest votes is eliminated in the usual STV fashion; these votes now need to be redistributed. For each vote that is to be redistributed, the eliminated first preference is removed from the head of the sequence, and moved to last place to make the length of the sequence the same as before. (Since this is now in position k in the sequence and there are only $k - 1$ candidates remaining, this element will never be called upon to be decrypted. Its only purpose is to provide padding so that no-one can tell from the length of the sequence how many times this vote has been transferred.) The new first place in the list, representing the second choice by the voter, is now decrypted. If Esther is the eliminated candidate then our vote becomes as depicted in Figure 14. The cells shaded in light grey, as before, show encrypted onions that no-one can read; the cell shaded in dark grey shows an encrypted onion that has been moved from the top to the bottom, and whose only remaining purpose is to serve as padding.

These votes are now added to the appropriate piles; in the case of our example ballot form, the vote will be added to Duncan’s pile. Any pile that has been added to is now

Duncan
$[[\text{Bob}]]_{PK_T}$
$[[\text{STOP}]]_{PK_T}$
$[[\text{Charlie}]]_{PK_T}$
$[[\text{Alice}]]_{PK_T}$
$[[\text{Esther}]]_{PK_T}$

Figure 14. Eliminating Esther

STOP
$[[\text{Charlie}]]_{PK_T}$
$[[\text{Alice}]]_{PK_T}$
$[[\text{Esther}]]_{PK_T}$
$[[\text{Duncan}]]_{PK_T}$
$[[\text{Bob}]]_{PK_T}$

Figure 15. Running out of preferences

completely remixed: each component of each vote is re-encrypted, and the whole pile is mixed. It is now impossible for anyone to distinguish first preferences from transferred second preferences. The only visible part of our ballot form is Duncan's name at the top, and the onions have been re-encrypted so no-one can pattern-match on them; this ballot form cannot now be distinguished from any of the others in the pile.

The process continues, with elimination of the candidate now holding the smallest pile. At each stage, transferred votes are thoroughly mixed into the pile, so that votes cannot be traced. The mixing process is exactly the same as happens before the tallying process, and it is audited in exactly the same way.

When a 'stop' onion is decrypted, that vote is then transferred to the pile of discarded votes, and it takes no further part in the tally. Our example ballot form will reach this stage after the elimination (in some order) of Esther, Duncan and Bob; it will then read as the ballot form in Figure 15.

From the point of the second elimination onwards, it is possible that when the new top preference is decrypted it turns out to be a vote for a candidate that has already been eliminated. The (usual) STV semantics require that this eliminated candidate cannot receive the vote, and so the next preference needs to be looked at; the procedure here is to collect together into a single pile all of the votes that are now for already eliminated candidates, remove the head of the list, move it to the back to pad it to the proper length again, and perform an anonymous re-encryption mix

before decrypting the next preference down. This prevents the problem of revealing several preferences in a row from the same ballot form if the next few preferences are all for eliminated candidates. This anonymous re-encryption mix is naturally audited in the same way as the mix process that comes before the tallying stage.

4.3.6 Analysis

The advantages of operating in this fashion are considerable. We look at each of the problems of Section 4.2.2 in turn.

Unused preferences. These simply never get decrypted at all, meaning that they cannot be used for covert channels. Once a candidate has been elected, the tallying process stops; this means that no preferences below the winning candidate will ever be decrypted. The lazy decryption ensures that a preference always stays hidden until it is needed for the count.

Transfer history. Italian-style attacks are rendered impossible because the whole decrypted vote is never seen: we see that at the first stage some votes are transferred from A to B, and that at the second stage some are transferred from B to C, but we never discover how many (if any at all) are transferred from A to B and then to C. The transfer history is never disclosed to anyone.

Absolute rankings. We also never learn the absolute ranking of the decrypted votes (except when required by the STV semantics—so, for example, we learn that the first preferences are indeed first preferences because they are decrypted before any transfers have taken place). We cannot tell, when a vote is transferred from A to B, how many of the previously eliminated candidates were listed above A on this ballot form.

There is one further advantage to this scheme over the FPTP re-encryption mix *Prêt à Voter*. With FPTP, and random index values in the onions, it was necessary to ensure that the initial seed values were drawn from a particular binomial distribution in order to make the tallying tractable. There are no such issues here. Treating each ranking as a separate onion renders this unnecessary. (It should be noted that work by Ryan [7] written concurrently with this paper solves this for FPTP *Prêt à Voter* by using Paillier encryption rather than ElGamal.)

One minor problem remains: it is still possible to tell from the receipt how many preferences were indicated on the ballot form. If the system allows zero preferences to be made, effectively making this a spoilt ballot, it is possible for a coercer to force particular voters unsympathetic to his cause to lose their votes, by insisting that they come out of

the booth with a blank receipt voting. This can be solved, at the expense of a minor complication of the voting process. For each ballot paper, an extra ‘candidate’ is included on the form, in a random place, labelled ‘STOP COUNTING HERE’ or some such. The ‘stop’ onion should then be located at this position in the ballot form’s list of onions rather than appearing as an extra onion at the end. The semantics of creating a vote are now the same as before, with onions being added to the vote sequence in order of the candidates they represent, but after the ‘stop’ onion is added, the rest of the preferences can be filled in arbitrarily, because they will never be decrypted. A receipt can thus be produced that contains a full ranking regardless of how many candidates the voter wanted to indicate. The voter will know in what position the ‘stop’ candidate appeared, and so will know how many preferences will be included in the count; but a third party will not know this and so will not be able to distinguish a spoilt ballot (a ‘1’ next to the ‘stop’ onion) from a fully ranked ballot (with the ‘stop’ onion in last place).

There are two ways to achieve this. One is to allow the booth device to see where the ‘stop’ onion is so that the voter can indicate a partial ranking and the booth can add the ‘stop’ onion and the remaining places. (If this route is followed, a deterministic method of filling the remaining places may be required to avoid the possibility of a covert channel emanating from the booth machine. The simplest option is to require the booth to fill in the remaining places from top to bottom; since the candidate list has been randomised, this will still result in random padding of the list.) This may be regarded as unsatisfactory because one of the design goals of Prêt à Voter is that the booth machine should know nothing about the vote, and this would ideally include how many candidates have been indicated.

The other method, of course, is to insist on the voter filling in a complete ranking, including the ‘stop’ onion and any remaining places. This gives the voter a way to specify a partial ranking, but now the booth machine does not need to know the position of the ‘stop’ onion, and the receipt bears no information about how many candidates were indicated. This is much to be preferred if the burden on the poor voter is not deemed too great.

Forcing the voter to complete the whole sheet is not considered problematic in Australia, where voting is compulsory and a complete ranking must be specified. Australia has no notion of a ‘stop’ onion, so here the voter is required to specify preferences that might affect the result of the election. Disaffected and disinterested Australian voters often fulfil their statutory obligation by simply filling in the boxes down the page in increasing order, starting with a ‘1’ at the top; it is to minimise the effect of these ‘donkey votes’ that the candidate names appear in a random order on the form.

4.3.7 STV elections with multiple seats

This paper has concentrated entirely on STV elections in which only a single winner is returned. The reason for this is that the usual STV semantics for elections to fill more than one seat involves transferring fractions of a vote. This is not easy to achieve without making it rather harder to hide the transfer history of a ballot form.

When only a single seat is to be filled, a candidate is declared the winner as soon as he has more than half of the votes. The *quota* for declaring a candidate elected is thus the smallest integer strictly greater than half of the total number of votes cast. Here, the quota serves only as a means for determining when one may stop tallying because the future course of vote transfers could not affect the result.

With more than one seat, there are various methods of calculating what the quota should be, but typically for an n -seat election with w votes cast the quota will be just over $\frac{w}{n}$ votes. The procedure when someone hits quota is rather more complicated than with a single seat election.

The STV principle is that votes should not be wasted: if a vote cannot be used for one candidate because that candidate has been eliminated, the vote should be transferred elsewhere. This equally applies to votes that exceed the quota: the *vote surplus*, the votes that a candidate has attracted beyond the minimum required to elect him, should also be transferred elsewhere. If this were not done, there would be considerable scope for tactical voting, because one might consider that one’s favoured candidate was all but certain to be elected, and that one might therefore be better off voting for someone else to avoid one’s vote being ‘stuck’ on the favoured candidate’s unnecessarily large pile of votes.

Transferring the surplus is not a case of simply transferring the last few votes to reach the elected candidate, however. All votes in the pile should be considered equally suitable for transfer. What is usually done is to transfer a fraction of each vote in the pile to its next indicated candidate, so that every vote is partially transferred and the total remaining exactly meets the quota. If the quota is 34 votes and the candidate has 47 votes, then the vote surplus is 13 votes, and so $\frac{13}{47}$ of each of the 47 votes is transferred. In a manual and paper-based count, the value of the vote (in this case, $\frac{13}{47}$) would be noted on the top of the ballot form, and the form would be moved to the next pile down.

With each occurrence of a candidate hitting quota, the arithmetic can become more and more convoluted. When the second candidate achieves quota, his vote count will include many ‘whole’ votes but may also include some of the ballot forms worth only $\frac{13}{47}$ of a vote. If this candidate has $36\frac{13}{47}$ votes, his vote surplus is $2\frac{13}{47}$; we must therefore transfer $2\frac{13}{47} / 36\frac{13}{47} = \frac{107}{1705}$ of each vote. Some of these transferred votes were whole votes, so are now worth $\frac{107}{1705}$; some, however, were worth only $\frac{13}{47}$ of a vote, so are now

worth $\frac{13}{47} \cdot \frac{107}{1705} = \frac{1391}{80135}$.

It is, of course, not difficult to keep track of this algorithmically and to ensure that the computer-based tallying is free from the errors that (as one might well imagine) plague STV manual paper counts. The problem is that the value of the vote needs to be kept along with the vote itself, and this means that some information can be gained about the transfer history of a vote. Since usually the vote surpluses for each candidate will turn out to be different, one can tell, from looking at a vote's remaining value, which surplus transfers it has been involved in so far.

One might well be prepared to live with this. The information gained by someone watching the tallying is not much, and it is still extremely unlikely that anyone will be able to reconstruct enough of a decrypted ballot form to be of any use.

If not, there is a way to avoid the information leak, though it comes at a price. When the first candidate to be elected has a surplus of s over a quota of q , we need to transfer $\frac{s}{s+q}$ of each of his votes. Rather than split the vote by duplicating it and noting that the one transferred is worth $\frac{s}{s+q}$ and the one left for the elected candidate is worth $\frac{q}{s+q}$, we can start by replicating *every* vote in the whole election $s+q-1$ times, so that each vote is now a collection of $s+q$ votes. For the votes being partially transferred, we transfer s of the replicated votes and leave q behind. (That the votes are copies does not matter: the next stage is to perform an anonymous mix on each pile, and after that we will no longer be able to tell which votes are copies of which.) We also scale up the quota by $s+q$. Now every vote is a whole vote again, and we proceed as normal until the elimination of the next candidate, after which we will end up scaling the votes again. This continues until the final candidate is elected.

This approach completely solves the problem of the information leak: we are back to a situation where only the bare minimum of information needed for the tallying is revealed. The only awkwardness is that the number of votes in the system gets big very quickly. With a large electorate, this method could cope with a small number of seats, but the data storage would rapidly get out of hand.

Improving on this is the subject of current research. A forthcoming paper will detail a rather better solution that keeps the vote values secret but avoids the vote replication.

5. Conclusions

In this paper, we have shown how single transferable votes may be elegantly and efficiently incorporated into Prêt à Voter. The size of the vote is linear in the number of candidates, so there is no significant strain on resources in terms of network activity. More importantly, the lazy decryption semantics ensures that no information is uncovered about

the vote beyond what is necessary to complete the tallying, and the anonymous mix at each stage of the tallying prevents tracing of a ballot form through the tallying process so that even the preferences that are decrypted cannot be used to recreate whole decrypted ballot forms.

In addition, we retain all the advantages of the re-encryption mix incarnation of Prêt à Voter, including the distributed ballot creation, the on-demand printing of ballot forms, and the ability to run each audit as many times as is required; we also avoid the awkwardness that the FPTP version has of requiring the initial index values inside the onion to be drawn from a particular probability distribution.

The nature of the Prêt à Voter ballot form lends itself to being extended to cover voting systems requiring a full ranking; the question this paper answers is one of how to rework the underlying cryptography. Some other e-voting systems would be much harder to adapt to STV right from the start, because the physical process of casting a ballot is not designed to support it. Adapting Punchscan to support ranking nine candidates, for instance, would seem to require setting up a nine-by-nine grid on the ballot form², so that the voter can select one letter in the first row, one in the second, and so on; this would work but might be beyond the wit of many voters to complete. It is not clear how to adapt Three-Ballot without scaling up the number of ballot forms that need to be filled in; voters might have difficulty in completing the TwentySevenBallot voting process correctly.

Acknowledgements

Peter Ryan has been particularly helpful during the course of this research in being able and willing to thrash ideas out and sort the wheat from the chaff.

This paper also owes much to the many useful discussions on the themes represented here with David Lundin, Roger Peel, Steve Schneider and Zhe Xia (Joson), who (with me) form the E-voting Research Group at the University of Surrey. David and Joson are PhD students working in related aspects of e-voting. The group as a whole is currently implementing STV using the form of Prêt à Voter involving RSA onions, for the forthcoming University of Surrey Student Union election as part of an entry to VoComp³. I am grateful to them all for giving me several people on whom to blame the mistakes in this paper.

References

- [1] D. Chaum, R. Carback, J. Clark, A. Essex, K. Fisher, B. Hosp, S. Popoveniuc, and J. Robin. Punchscan and VoComp. Presented at the Rump Session of Crypto 06, 2006.

²or giving the voter nine different colours of pen!

³See <http://www.vocomp.org/> for details.

- [2] D. Chaum, P. Y. A. Ryan, and S. A. Schneider. A Practical Voter-Verifiable Election Scheme. In *10th European Symposium On Research In Computer Security (ESORICS 2005)*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139. Springer-Verlag, 2005.
- [3] M. R. Clarkson and A. C. Myers. Coercion-Resistant Remote Voting using Decryption Mixes. In *Workshop on Frontiers in Electronic Elections (FEE 2005)*, 2005.
- [4] M. Gomulkiewicz, M. Klonowski, and M. Kutylowski. Rapid Mixing and Security of Chaum's Visual Electronic Voting. In *Proceedings of ESORICS 2003*, Oct. 2003.
- [5] S. Popoveniuc and B. Hosp. An Introduction to Punchscan. In *IAVoSS Workshop On Trustworthy Elections (WOTE 2006)*, July 2006.
- [6] R. L. Rivest. The ThreeBallot Voting System. Available online at <http://theory.lcs.mit.edu/~rivest/Rivest-TheThreeBallotVotingSystem.pdf>, Oct. 2006.
- [7] P. Y. A. Ryan. Prêt à Voter with Paillier Encryption. Technical Report 1014, Newcastle University, School of Computing Science, Apr 2007.
- [8] P. Y. A. Ryan and S. A. Schneider. Prêt à Voter with Re-encryption Mixes. In *11th European Symposium On Research In Computer Security (ESORICS 2006)*, volume 4189 of *Lecture Notes in Computer Science*, pages 313–326. Springer-Verlag, 2006.