Faster Print on Demand for Prêt à Voter

CHRIS CULNANE, University of Surrey
JAMES HEATHER, University of Surrey
RUI JOAQUIM, University of Luxembourg
PETER Y. A. RYAN, University of Luxembourg
STEVE SCHNEIDER, University of Surrey
VANESSA TEAGUE, The University of Melbourne

Printing Prêt à Voter ballots on demand is desirable both for convenience and security. It allows a polling station to serve numerous different ballots, and it avoids many problems associated with the custody of the printouts. This paper describes a new proposal for printing Prêt à Voter ballots on demand. The emphasis is on computational efficiency suitable for real elections, and on very general ballot types.

1. INTRODUCTION

There are several projects on end-to-end verifiable election protocols either already used, or intended to be used in government elections, including the Scantegrity II project in Takoma Park [Carback et al. 2010], and the StarVote project in Texas [Benaloh et al. 2011]. Various other e2e systems such as VoteBox [Sandler et al. 2008], Wombat [Rosen et al. 2012] and Helios [Adida 2008] have been used for non-government elections. Many other proposals exist in the academic literature.

Our motivation is adapting Prêt à Voter for elections in the Australian state of Victoria for the 2014 state election. This requires STV[1] ballots with approximately 30 candidates, and various other ballot types. Although the proposed algorithm is specific to the Victorian project, it would work just as well for other versions of Prêt à Voter, including those that are filled in manually.

This paper introduces a new protocol for the verifiable printing on demand of Prêt à Voter ballot forms. A similar approach would work for any scheme in which voters select or arrange ciphers from a pre-generated ballot form. Scantegrity is the obvious other application. The crucial requirement is that voters (and others) can audit some ballot forms for correctness *without compromising voter privacy*, because the audit occurs before the person votes. Voters then vote on ballot forms that have not been audited.

The integrity of Prêt à Voter depends crucially on proper construction of the printed ballot forms, meaning that the plaintext candidate list that the voter sees must match the encrypted values for that ballot. Consequently this proposal details opportunities for auditing their correct construction and printing. The proposed construction is very computationally efficient and retains most of the desirable properties of existing print-on-demand proposals in the literature. The information flow of our scheme is similar to Markpledge 2 [Adida and Neff 2009], though the auditing is different. The main idea is that the device encrypts the vote directly using randomness generated by others.

Also the system must address the question of "kleptographic" privacy attacks [Gogolewski et al 2006], in which the (public) ciphertexts contain deliberately poorly-chosen randomness that exposes the vote. This is possible whenever the entity building the ciphertexts also controls all the randomness used. This problem is addressed by distributing the process of inputting randomness into ciphertexts.

In summary, distributed ballot generation is a desirable property to have for the following reasons:

— ensuring the candidate lists are randomly generated,
— ensuring no single generating entity knows all the (plaintext) candidate lists, and

[1]Single Transferable Vote (STV) is a proportional representation system that allows voters to number many candidates in their order of preference.

This work is supported by the UK Engineering and Physical Sciences Research Council, under grant EP/G025797/1, and by Fonds National de la Recherche Luxembourg, under the SeRTVS project.

— ensuring extra information about the candidate list can't be leaked in the ballot ciphertexts (as in kleptographic attacks).

This proposal is designed so that the entity that builds the ciphertexts (the printer) has a deterministic algorithm to follow. This provides a way to distribute the expensive cryptographic operations to the network of printers, whilst retaining the central, distributed, generation of randomness to maintain the three properties above.

1.1. Protocol overview

Our protocol has two roles. The "randomness generation servers," of which a threshold are trusted for privacy, send randomness to a "printer". The "printer" uses only that randomness to generate the ballots, which it can then print on demand. In brief:

(1) Each randomness generation server generates some randomness, commits to it publicly, and sends the opening secretly to the printer.
(2) The printer uses that randomness to generate the encrypted ballot, which it publishes.
(3) When required, the printer prints the next ballot in sequence, with human-readable candidate names.

There are thus two important points for public auditing:

(1) An audit of the encrypted ballot produced in step 2, to check that the candidate ciphertexts are valid and that the printer used the proper randomness. This is described in Section 2.5.
(2) A standard Prêt à Voter audit of the printed ballot from step 3, to check that the printed human-readable candidate names match those of the encrypted ballot. This is described for our scheme in Section 3.2.

1.2. Prior work and our contribution

There are several papers that provide constructions for distributed ballot generation in Prêt à Voter [Ryan and Schneider 2006; Ryan 2007; Ramchen and Teague 2010]. However, most of these do not extend to IRV/STV ballots, and those that do require a substantial amount of cryptographic computation. Our motivating application for Victorian state elections requires approximately 60 ciphertexts to be generated, permuted, and committed for each ballot.[2] These ciphertexts must also be securely obtained by printers in polling stations. The computational cost of generating a sufficiently large number of ballots using existing methods (such as [Ramchen and Teague 2010]), is too high.

In our scheme, deviations from proper ballot generation will be detected with bounded probability by audit. This is weaker than the pure cryptographic approaches, which involve a zero knowledge proof from which deviations are detected with overwhelming probability. Our approach allows auditing to make the error bound arbitrarily small, but it could require a lot of auditing. This seems a reasonable tradeoff given the computational efficiency gain, since improper ballot *printing* must be detected by random audits anyway. See Section 6.4 for more discussion of this issue.

Although this approach would work for simpler ballot types, the tradeoff is only justifiable if the cryptographic approaches that have an overwhelming probability of detecting cheating are infeasible. For first past the post and other simple voting methods, existing protocols in the literature would be quite feasible.

It seems likely that our techniques would extend to schemes such as Wombat, StarVote, VoteBox and Helios, in which the device encrypts the vote directly. The idea would be to extend the "challenge" process available in those systems to include a check that the device had used the right randomness. However, without more careful analysis of each particular scheme it is difficult to be certain that this would completely remove all opportunities for kleptographic attacks. Nor would this proposal necessarily represent the right tradeoff between efficiency and security for those schemes.

[2]This includes the 30-candidate STV list and two other voting methods each involving approximately 15 candidates.

Our approach has similar aims to Backes *et al.*'s privacy preserving accountable computation [Backes et al. 2013] and to Verifiable Random Functions [Micali et al. 1999]. However, those works have a single trusted generator of the randomness, which we need to avoid, though in both cases it might be possible to generate the secret information in a distributed way and then publish the public information. Also, both constructions requires a zero knowledge proof to be generated for every computation, which is not efficient enough for our application.

Throughout this document when we refer to the "Printer" we are in fact referring to the tablet device that is connected to the printer. As such, the "Printer" has the processing power you would expect to find on a mid-range tablet. The "Web Bulletin Board (WBB)" is an authenticated broadcast channel with memory. It also returns a signature on messages posted on it, and performs some basic tests for correctness before accepting posts.

The next section describes how ballots are generated and how the generation is audited. Section 3 describes how they are printed on demand and how printing is audited. Section 4 descibes an optimisation. Section 5 describes some practical details related to runtime failures. Section 6 gives an informal security analysis.

2. BALLOT GENERATION

2.1. Overview

The main idea is that the printer generates a permuted list of candidate ciphers using randomness values generated by a distributed set of peers. As such the printer undertakes the expensive crypto operations, but does not have any influence over the values used in those operations. This prevents the printer from mounting kleptographic attacks or otherwise having any influence over the ciphertexts.

During ballot generation the printer is audited to ensure that it has performed honestly: if a sufficient number of ballots are audited and shown to be correct then we can gain a high assurance that the printer has behaved honestly. The definition of "honest" is different from standard versions of Prêt à Voter, in which a dishonest printer can only misalign the printed candidate names with the ballot onion. In our version, a dishonest printer may also attempt to generate invalid ballot ciphertexts or perform a kleptographic attack by using randomness other than that specified by the protocol. However, these two kinds of cheating can be detected by a ballot-generation audit—see Section 2.5.

Notation:

$Enc_k(m;r)$. is the encryption of message m with public key k and randomness r.[3]
$Dec_k(m)$. is a decryption of m using the private key k.
$ReEnc(\theta;r)$. is a re-encryption (re-randomisation) of the ciphertext θ using the randomness r (this abstracts the requirement of knowing the public key, which is a requirement for ElGamal re-encryption).
$c(m)$. is a perfectly hiding commitment to message m (using some randomness not explicitly given).
$c(m;r)$. is a perfectly hiding commitment to message m (using randomness r).[4]
PK_E. is the election public key (which is thresholded).
PK_P. is the printer's public key (which is not thresholded/distributed).
SK_P. is the printer's private key.
n. is the number of candidates.
b. is the number of ballots to be generated for each printer.
G. is the number of randomness generation servers.
$RGen_i$. is the i-th randomness generation server.
$SymmEnc_{sk}(m)$. is a symmetric-key encryption of m under the symmetric key sk.

[3]The Victorian project uses Elliptic Curve El Gamal.
[4]In the vVote project we use the hash-based commitment scheme described in [Jakobsson et al. 2002].

Candidate Name	ID
Vladimir Putin	$cand_1$
Mohamed Morsi	$cand_2$
...	...
Mahmoud Ahmadinejad	$cand_n$

Fig. 1. Initial Ballot Input: Candidate Identifiers

PrinterA:1	...	PrinterB:1	...	PrinterC:1
PrinterA:2	...	PrinterB:2	...	PrinterC:2
⋮	⋮	⋮	⋮	⋮
PrinterA:b				

Fig. 2. Initial Ballot input: Serial numbers for printers A,B,C.

$h(m)$. is a cryptographic hash of the message m.[5]
$SymmDec_{sk}(m)$. is a symmetric-key decryption of m using key sk.

We will post on the public WBB values that are encrypted with a threshold key, or perfectly hiding commitments. We will not post values that are encrypted with a non-thresholded key. We could have used computationally hiding commitments or encryptions with non-thresholded keys, but either of these would have meant that a single leak of relevant parameters, even quite a long time in the future, could have been combined with WBB data to violate ballot privacy. Our system does not achieve everlasting privacy, but it achieves a somewhat related weaker property, that no single entity's data (apart from the printer's) is enough to break ballot privacy, even given WBB data.

2.2. Pre-Ballot Generation

Before the ballot generation starts the following must occur:

(1) The election public key sharers jointly run a distributed key generation protocol to generate a thresholded private key and joint public key PK_E.[6]
(2) A list of candidate identifiers is generated and posted on the public WBB, as shown in Figure 1. Candidate identifiers are arbitrary, distinct elements of the message space of the encryption function.[7]
(3) For each printer, a list of serial numbers of the form "PrinterID:index" is deterministically generated and posted on the public WBB. This serial number is just the literal string as given. These serve as row indices for later computation.
(4) Each printer constructs a key pair and publishes the public key PK_P.

All this data, which is posted on the WBB immediately before ballot generation, is shown in Figures 1 and 2.

The following sections describe the process of ballot generation for a single printer, but it should be clear how the same process will be run in parallel for each printer.

[5]We use 256-bit AES and SHA-256 respectively. This means that the computational difficulty of guessing a key (2^{256}) is much greater than that of finding a collision (2^{128}). However, this seems justifiable since collision-finding is only useful for cheating during ballot generation, which must be performed in a restricted time, while guessing the symmetric key can be used to break ballot privacy long after the election.
[6]We keep the key sharers and the randomness generation servers conceptually separate, even if we end up using the same servers.
[7]For vVote, candidate identifiers are elliptic curve points either randomly selected or calculated as part of the optimisation used to speed up mixing, depending on the type of race.

Serial No	Encrypted Randomness		
PrinterA:1	$SymmEnc_{sk_i}(r_{1,1}\|\|R_{1,1})$	...	$SymmEnc_{sk_i}(r_{1,n}\|\|R_{1,n})$
PrinterA:2	$SymmEnc_{sk_i}(r_{2,1}\|\|R_{2,1})$	...	$SymmEnc_{sk_i}(r_{2,n}\|\|R_{2,n})$
⋮	⋮	⋮	⋮
PrinterA:b	$SymmEnc_{sk_i}(r_{b,1}\|\|R_{b,1})$	...	$SymmEnc_{sk_i}(r_{b,n}\|\|R_{b,n})$

Fig. 3. Ballot Input: Table RT_i, sent privately from peer i to printer A without public posting.

Serial No	Committed Randomness		
PrinterA:1	$c(r_{1,1};R_{1,1})$	...	$c(r_{1,n};R_{1,n})$
PrinterA:2	$c(r_{2,1};R_{2,1})$	...	$c(r_{2,n};R_{2,n})$
⋮	⋮	⋮	⋮
PrinterA:b	$c(r_{b,1};R_{b,1})$	...	$c(r_{b,n};R_{b,n})$

Fig. 4. Commitment to Initial Ballot Input: Table CRT_i, posted by peer i on the WBB.

2.3. Randomness Generation

The randomness generation consists of each server $RGen_i$ generating a large table of secret random values and sending them (privately) to the printer after posting (public) commitments to them on the WBB.

2.3.1. Notation

RT_i. is $RGen_i$'s private table of encrypted random values (Fig 3).
CRT_i. is $RGen_i$'s public table of commitments to the values in RT_i (Fig 4).

2.3.2. Detailed algorithm. In detail: each random value has length k, where k is a security parameter which should be about 256 bits. Each server $RGen_i$ generates a random symmetric key sk_i. It then generates a table of $b*n$ pairs of pieces of random data, each of size $2*k$ bits and encrypted under sk_i. The table is denoted by RT_i, and each pair can be retrieved by the serial number and column, or the row and column.[8] The result is shown in Figure 3. The idea is that the first element of each pair, $r_{(row,col)}$, will be used later by the printer; the second element, $R_{(row,col)}$, is used to commit to $r_{(row,col)}$ and to open the commitment in case of audits.

Each server commits to $r_{(row,col)}$ by posting on the WBB a commitment to it using randomness $R_{(row,col)}$. The table of commitments is shown in Figure 4. Call the table CRT_i. Each peer $RGen_i$ posts its CRT_i, and checks all CRT_i's are posted before sending RT_i privately to the printer.[9] $RGen_i$ also encrypts sk_i with the printer's public key and sends the result (denoted $esk_i = Enc_{PK_P}(sk_i;r)$) to the printer.

2.4. Ballot Permutation and Commitment

The printer receives the respective RT_i and corresponding (encrypted) key esk_i from each server. The printer also downloads or constructs the candidate identifiers and serial numbers shown in Figures 1 and 2. The printer now needs to encrypt and permute the candidate identifiers.

For each candidate identifier $cand_k$ in the pre-committed table in Figure 1 it encrypts it using the combined randomness from the RT tables, received from the randomness generation servers. To combine the randomness the printer first decrypts the encrypted symmetric key esk_i received from

[8] For example $RGen_i.RT_i(PrinterA:1,2)$ and $RGen_i.RT_i(1,2)$ will return the pair of encrypted random values in the second column for the first ballot: $SymmEnc_{sk_i}(r_{1,2}||R_{1,2})$.

[9] This is to stop the last peer choosing their randomness when they know the others'. If this was not enforced, then one bad randomness generation server colluding with a printer could determine the randomness values for each of that printer's ballots, thus breaking privacy. The bad server would wait until the printer told it all the other servers' random values, then generate its own to produce a particular final value.

each server and then uses the resulting key sk_i to decrypt the randomness in RT_i. For each element of each RT, the printer checks that the decrypted pair $r_{row,col}, R_{row,col}$ opens the commitment at $CRT_{row,col}$. It challenges any that do not—see Section 2.4.1 for this case.

The decrypted first elements $r_{row,col}$ from each peer are concatenated and hashed. The printer then uses the hash output for randomness when encrypting the candidate identifier under PK_E. The resulting ciphers are then sorted into canonical order to produce a random permutation. These ciphers are posted on the public WBB. Note that the output of the encryption is pseudo-random and as such sorting the encrypted ciphers will give a pseudo-random permutation π. The printer retains this permutation so that it can print the plaintexts in the appropriate order when requested to print that ballot. After the ciphers are submitted to the WBB, the audit protocol detailed in Section 3.2 can be run. The algorithm run by the printer is given in Algorithm 1.

Algorithm 1: Deterministic Encryption by Printer

for $i = 1 \to G$ **do** ▷ Decrypt the symmetric keys from the *RGen*Servers
  $sk_i \leftarrow Dec_{SK_P}(esk_i)$
end for
for $j = 1 \to b$ **do** ▷ b is the number of ballots.
  for $k = 1 \to n$ **do** ▷ n is the number of candidates.
    $rand \leftarrow SHA(SymmDec_{sk_1}(RT_1(j,k)) \| \ldots \| SymmDec_{sk_G}(RT_G(j,k)))$
    $CT_{j,k} \leftarrow Enc_{PK_E}(cand_k; rand)$
  end for
  $CT_{(j)} \leftarrow Sort(CT_{(j)})$ ▷ $CT_{(j)}$ returns the entire row
  $perms_j \leftarrow \pi$ ▷ π is the permutation applied to sort CT_j.
end for
Send CT to WBB.

The intention is that only the printer knows which ciphertexts correspond to which candidates, but its algorithm for generating those ciphertexts is deterministic. Hence it cannot use the ciphertexts to leak information without detection. Of course, the printer could always leak that information via a side channel, but this is unavoidable and occurs with any form of electronic ballot printing or marking.

2.4.1. What the printer should do if $RGen_i$ cheats. It's important in the above protocol that the printer checks the opening of each commitment, *i.e.* checks that for each element of each RT, the decrypted pair $r_{row,col}, R_{row,col}$ opens the commitment at $CRT_{row,col}$. It is important in practice that the printer raise an alarm on any commitments that are not correctly opened. Exactly how such a dispute should be resolved requires some careful engineering of procedures. It is difficult to tell whether the printer or the randomness generation server is misbehaving without exposing private ballot data. This is not necessarily a problem, because the randomness contributed by other randomness generation servers would not be exposed. Hence the other ballots remain private.

Note that the issue does not affect public verifiability, because the absence of proper commitment opening would be detected by an audit of this ballot. It does, however, affect accountability: if an audit detects that the value used to encrypt a ballot was not a valid opening of the commitment on the WBB, we would like to know whether it was the randomness generation server or the printer that cheated. If we insist that the printer performs this check, then we can be certain that a failed audit is the printer's fault.

2.4.2. Alternative construction with PRNGs. Rather than generate a separate random value for each candidate in each ballot, an alternative is to generate one random value for each ballot, then use a cryptographic PRNG to expand it to produce randomness for all of the encrypted values in the ballot. This introduces an assumption on the good expanding behaviour of the PRNG, but substantially reduces the communication costs of the protocol. Hence the tables of Figures 3 and 4 need only 2 columns rather than $n+1$. It does not significantly change auditing.

However, we have not taken this approach for the Victorian system due to practical concerns about PRNG implementations. The most important is the possibility that variations in implementations could imply a failure of reproducibility of the random sequences, which is required for audit. In principle this shouldn't be a difficult issue to address. In practice we were concerned this would make it significantly harder to write an independent verifier, so we opted to omit the PRNGs.

2.5. Ballot Generation Auditing

Someone chooses a suitable percentage of the ballots to audit at random.[10] For each ballot selected, the printer posts on the WBB the randomness it used during the generation, *i.e.* to open the commitments for that SerialNo in each peer's *CRT* table. The printer can either have this stored, or else can recalculate it from the encryptions it received prior to ballot generation. Anyone can verify the commitment openings $(r_{row,col}, R_{row,col})$ and reconstruct the ballot ciphertexts from them. Thus anyone can check that the ballots were correctly constructed and that the printer used the appropriate randomness.

3. PRINT ON DEMAND

This section describes what happens when a voter appears at a polling place. The printer needs to print a pre-generated ballot for the appropriate district. The printer knows the plaintexts and permutation for a particular ballot so can easily print the appropriate ballot out.

However, there is a risk that a misbehaving printer might print a completely invalid ballot, *i.e.* one that hasn't been part of the generation process described above. Although this is detectable by audit, we prefer for practical reasons to prevent it altogether. Hence we require that the printer obtain a signature from the WBB in order to create an authentic ballot. The WBB is attesting to those ciphertexts matching what the printer has already committed to.

The vVote project uses a combined EBM and scanner rather than the traditional Prêt à Voter technique of filling in the ballot with a pencil and then scanning it. The print on demand protocol works exactly the same either way, so the description below is intended to work for either the traditional pencil and scanner setup, or the vVote-style combined scanner and EBM.

Figure 5 shows the message sequence chart for print on demand, which is elucidated in Section 3.1 below. The authorisation, ballot reduction and serial number signing all take place together in a single round of communication. The signatures generated are deterministic BLS signatures [Boneh et al. 2004], including the signature from the WBB.

3.1. Ballot Reduction

It is unpredictable exactly how many of each ballot will be required at each location. We could generate an abundant oversupply of ballots with exactly the right number of candidates for each division, but this could be quite expensive. Instead it is probably more efficient to generate an abundant oversupply of generic ballots with a larger than necessary number of candidates, then reduce it down to the appropriately sized ballot for the district/region it is going to be used in. This allows great flexibility about who votes at what polling place—any voter can arrive anywhere and have a ballot produced to match their voting eligibility.

If the ballot contains more ciphers than candidates, we need to reduce the ballot in a manner that can be verified. The following proposal has the nice feature that the voter (or the EBM if there is one) does not need to know where the blanks are in order to cast the vote: they just get a candidate list in the order that the ciphertexts for the candidates they need appear on the ballot. Then the EBM or scanner prints, and the voter checks, the voter's preference numbers against that order.

Suppose from now on there are m candidates in the division and n $(> m)$ ciphertexts on the ballot. ($n = m$ is a special case of the steps below.) The printer is supposed to use the ciphertexts for $cand_1, cand_2, \ldots, cand_m$. Of course it could cheat and attempt to use other ciphertexts instead,

[10] The questions of who chooses, how they choose, and how it can be guaranteed that they choose well enough to engender confidence in a particular election result are beyond the scope of this paper.

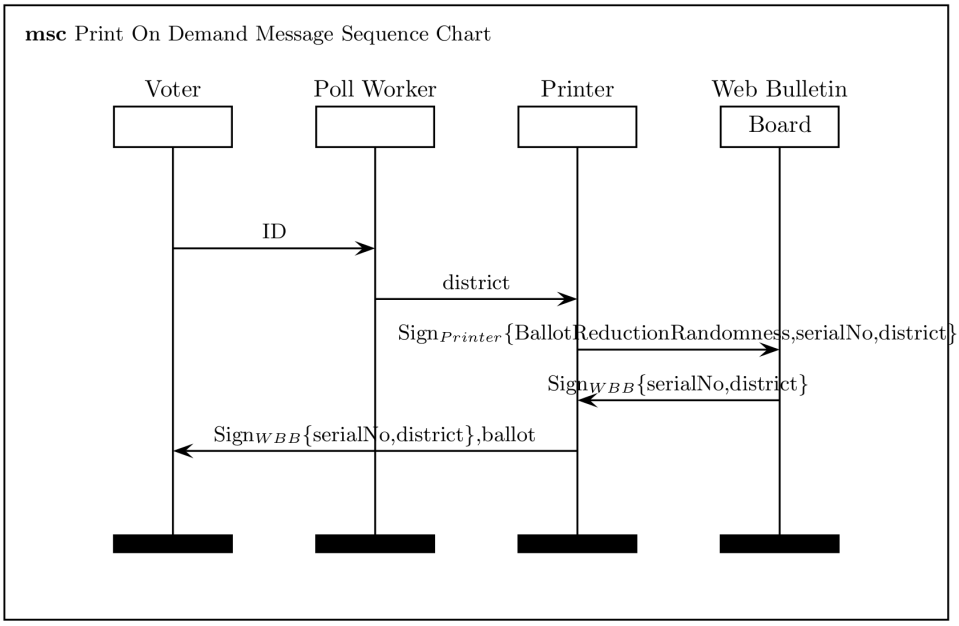

Fig. 5. Print on Demand Message Sequence Chart

but this could be detected at audit time like any other kind of bad printing. We want to be able to demonstrate afterwards on the WBB that it used the right ciphertexts. The protocol is as follows:

Pollworker:. authenticates the voters (using whatever secure or insecure method is traditional) and sends a print request to the Print On Demand device specifying the district/region they can vote in,[11]

Printer:. retrieves the next available ballot and looks up the number m of candidates in the submitted district/region.

Printer:. sends to the WBB:

— the serial number,

— the division, and

— a list *BallotReductionRandomness* of randomness values for the unused ciphertexts (i.e. the ones from $cand_{m+1}$ to $cand_n$), together with their respective permuted locations so the WBB can check them. The randomness values are those computed by the printer in the algorithm in section 2.4)

WBB:. checks that the ciphers held for $cand_{m+1}$ to $cand_n$ are encryptions of the candidate IDs of the unused candidates for the specified division.

— if valid it signs the serial number and the division and returns it to the printer, and posts the randomness values to the WBB so they can be publicly checked;

— if invalid it returns an error message.

Printer:. Checks the WBB signature of the serial number and division and, if valid, prints the ballot and signature. The printer knows the permutation and plaintexts so does not need to do any crypto to print the ballot

Voter:. votes on the ballot exactly as if it had been generated for the right number of candidates,

[11] More generally, it is the pollworkers' responsibility to authenticate the voter and request the ballot(s) that the person is eligible to vote on.

Voter:. shreds the candidate list,
EBM (or scanner if there is one):. submits the ballot to the WBB exactly as if it had been generated for the right number of candidates,
WBB:. accepts (and signs) the ballot only if it is accompanied by a signed serial number and division
EBM (or scanner):. prints the sig on the receipt,
Voter:. (optionally) checks the sig, which covers only data visible to the voter.
Voter:. later checks their vote on the WBB. They only need to check the serial number and order of their preference numbers—the correct opening of the unused (too big) candidate numbers will be universally verifiable.

3.2. (Print) auditing

Suppose a voter wants to audit a printed ballot, i.e. to check that the printed candidate list matches the ciphertexts on the WBB. The following is performed:

Voter:. requests an audit from the same printer that printed their ballot,
Printer:. sends to the WBB the randomness to open the commitments to the randomness on each CRT_i used to generate the ballot in the ballot generation phase
WBB:. checks the serial number has not already been voted on or audited and if not, opens the commitments, reconstructs the ballot, computes the permutation π, posts all the data on the public WBB, and sends a jointly signed copy of π (or candidate names in permuted order) to the printer
Printer:. The printer prints the signature
Voter:. checks the signed order of candidates π against the order printed on the ballot (note, the permutation signed by the WBB should reflect any successful ballot reduction already performed).
Voter:. takes their audited ballot home and checks that the value provided on the WBB matches the candidate order that was signed.

Figure 6 shows the message sequence chart for auditing.

3.3. Forward Secrecy

The randomness held on a printer is sufficient to reconstruct the ballots, and hence reveal the candidate orderings. If a printer is stolen, the randomness it holds will expose the associated ballot forms. Hence it is desirable for a printer to delete the randomness for ballots on which votes have been cast, since there is a potential privacy breach.

However, when a printer prints a ballot it must allow for a print audit which will require it to open the commitment to the randomness. Therefore, after a printer has printed a ballot, it must retain this randomness for some period of time.

After the ballot has been used to cast a vote, an audit is not allowed and so the randomness no longer needs to be retained and can be deleted. Deletion can be triggered by a confirmation message to the printer, signed by the WBB, when a vote is cast. Alternatively a time limit can be set on an audit request following ballot printing, and the randomness can be deleted after that time if no audit request has been received.

Note that the encrypted randomness values and symmetric key are sent directly to the printer and not posted on the WBB. As such, there is no publicly available information that could be combined with a stolen, but previously honest, printer to reveal used ballots. The symmetric keys sk_i should be deleted from the printers after the RT_i tables have been decrypted.

4. A FASTER VARIANT WITH A SHORTER PERMUTATION COMMITMENT

Although the above protocol is quite efficient, it still requires the WBB to do a lot of computation to open all relevant commitments and reconstruct the ballot permutation each time a print audit occurs.

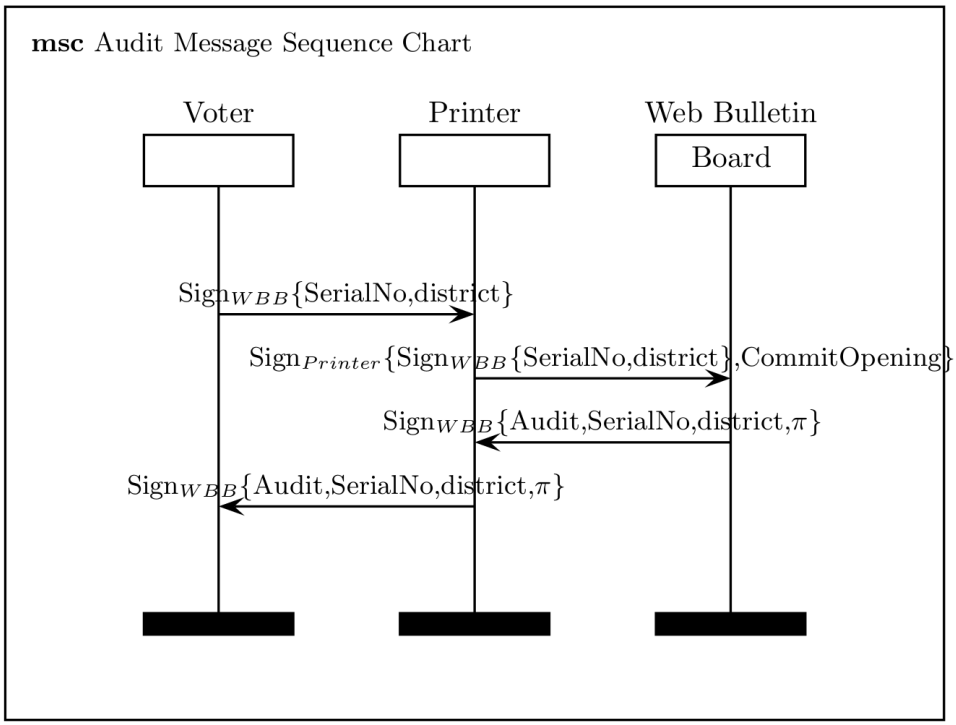

Fig. 6. Print on Demand Audit Message Sequence Chart

This is unfortunate because we would like to encourage ordinary voters to perform print audits by making them easy and fast.

One way to speed up print auditing is to ask the printer to commit to the permutation π directly when it generates the ballot, then ensure that this commitment to π is audited for proper generation (during ballot generation audits) and for conformance with the ballot permutation (during print audits). During a print audit, the WBB needs only to open and verify the commitment to π, then sign it and return it to the printer. Every print audit then triggers a ballot audit, which opens all the commitments just as described in Section 2.5, but this does not have to be done while the voter is waiting for the print audit to complete.

Like other randomness values used by the printer, the randomness used in the commitment must also be generated by the *RGen* servers, using a new column in each RT_i and CRT_i table. The printer retrieves the random value the same way that it retrieves all the others, and uses it to compute the commitment to π. That commitment is sent, along with ballot generation ciphers, to the WBB during the ballot generation stage. It is also audited, along with the ciphers, during the ballot generation audits so we gain a statistical assurance the commitments to the permutation are correct. During the print audit everything proceeds as described above, except the WBB only has to open and check the commitment and then sign the permutation based on that.

This does not affect universal verifiability, because the same data linking the ballot permutation to the commitments on the WBB is eventually published either way. However, the big advantage is it reduces the workload on the WBB during the critical time that the voter is waiting for the signature on π, since it can now defer the re-encryptions necessary to verify the permutation.

More precisely, if we add the required random values into the $n+1$-th column of each RT_i, Algorithm 1 would now be:

Algorithm 2: Deterministic Encryption by Printer with explicit WBB commitment to π.

for $i = 1 \rightarrow G$ **do** ▷ Decrypt the symmetric keys from the *RGen*Servers
  $sk_i \leftarrow Dec_{SK_P}(esk_i)$
end for
for $j = 1 \rightarrow b$ **do** ▷ b is the number of ballots.
  for $k = 1 \rightarrow n$ **do** ▷ n is the number of candidates.
    $rand \leftarrow SHA(SymmDec_{sk_1}(RT_1(j,k)) \| \ldots \| SymmDec_{sk_G}(RT_G(j,k)))$
    $CT_{j,k} \leftarrow Enc_{PK_E}(cand_k; rand)$
  end for
  $CT_{(j)} \leftarrow Sort(CT_{(j)})$ ▷ $CT_{(j)}$ returns the entire row
  $perms_j \leftarrow \pi$ ▷ π is the permutation applied to sort CT_j.
  $rand2 \leftarrow SHA(SymmDec_{sk_1}(RT_1(y,n+1)) \| \ldots \| SymmDec_{sk_G}(RT_G(y,n+1)))$
  $commit_\pi \leftarrow c(\pi; rand2)$
end for
Send CT and $commit_\pi$ to the WBB. to WBB.

Ballot generation audit (Section 2.5) would be exactly as above. Additionally, anyone can recompute the randomness value the printer used to commit to the candidate permutation π, and hence open that commitment and check that it matches the ballot permutation.

Print generation audit (Section 3.2) would be:

Voter:. returns to the same printer they previously used and requests an audit
Printer:. sends to the WBB the randomness to open the commitments to the randomness on each CRT_i used to generate the ballot in the ballot generation phase and the randomness to open the commitment to π.
WBB: (immediately). checks the serial number has not already been voted on or audited and if not, opens the commitment to π, checks it, and if valid sends a jointly signed copy of π (or candidate names in permuted order) to the printer.
WBB: (later). opens all the commitments for this ballot, reconstructs the ballot, computes the permutation π, posts all the data on the public WBB.

The final step of opening all the other commitments reduces the total number of audits that need to be done. If we relied entirely on the immediate check of the serial number and the frequency of ballot generation audits, then the system would still be universally verifiable, but the probability of the printer cheating successfully would be higher.

5. RUN-TIME FAILURES

5.1. Real-time Printer Replacement

If a printer is stolen or fails, all the ballots that had been generated by it are no longer available for use. This is important because the "printer" is a small tablet PC that would be easy to carry. Hence we need to have a suitable method for bringing replacement equipment back online during election time. For example, we could bring the randomness generation servers back online each night, or when needed, to generate new randomness, after having deleted those values they had already sent to printers. An alternative is for the randomness generation authorities to do the same thing as described above for a few extra as-yet-undeployed printers, put the data on the public WBB, and then ask each one to send their sk_i to some (distinct) entity who is going to be online at voting time.

5.2. Loss of WBB

The Print on Demand protocol requires the participation of the Web Bulletin Board. Intermittent loss of connection to the WBB will be handled by the communication infrastructure, but loss of the network or the WBB will require fallback to a mode of operation that is purely local.

Since vVote is intended to operate alongside standard paper ballots (for 2014) the fallback position will be to return to "plain EBM mode": the EBM machines will be used purely for constructing and printing a paper ballot which can then be cast with the standard paper ballots. No receipts will be issued and the vote will not be submitted through vVote. The Print on Demand service is not used at all for plain EBM voting.

6. SECURITY CLAIMS

This protocol is meant to achieve three classes of security properties:

> *integrity:* . meaning that all attempts to manipulate a voter's input into the mix would be detected by a ballot audit, print audit, the voter's check of their printed vote, the signature check, or the final check that their receipt was properly recorded on the WBB.[12][13]
> *privacy:* . meaning that either the printer itself or all of the randomness generating authorities must collude to reveal the contents of a vote.[14]
> *resistance to kleptographic attacks:* . meaning that a printer attempting to leak information via the WBB data will be detected with some probability.

6.1. Integrity based on audits

An informal argument for the integrity of each person's vote is:

— The ballot-generation audit confirms that the ballot is a permutation of properly-encrypted candidate identifiers.
— The ballot-printing audit confirms that the printed list of candidate names matches the encrypted candidate identifiers on the WBB.
— The voter's check of the EBM's printout confirms that the correct numbers (or other marks) are recorded against the correct candidate names.
— The signature check confirms that the printed number sequence matches what was submitted to the WBB.
— The check of the vote on the WBB confirms that the correct ciphertexts were used (in the case of a larger-than-necessary generated ballot) and that the vote submitted to the WBB was posted.

Of course the first two audits are performed only on ballots that are *not* subsequently voted on. The argument is that any attempt to manipulate the vote by generating or printing invalid votes will be detected by random audits with some probability.

6.2. Vote privacy

Clearly if the printer leaks its information it can violate vote privacy for everyone who used a ballot it printed. This means that practical opportunities for compromising the printer must be reduced as much as possible, *e.g.* turning off the wireless connection.

Apart from the printer and an electronic ballot marker (if there is one), no other single entity can violate vote privacy.

Claim: A collusion of all but two randomness generation authorities does not have sufficient information to recover the ballot permutation (in polynomial time with non-negligible probability).

---

[12] Of course this does not imply that they will always be detected, if those audits are not performed on the manipulated ballot. The claim is that any manipulation can in principle be detected by an audit if it is performed.

[13] A similar claim applies to attempts to manipulate the output of the vote mixing process which happens afterwards, but that is not the topic of this paper.

[14] It's also possible for a threshold of key sharers or a sufficient number of vote mixing servers to collude to reveal a vote, but that's not part of this sub-protocol.

Clearly if all the randomness generation authorities collude and share their information, they learn the contents of all ballots. If at least two choose their random values correctly and keep them secret until the others have committed, and if the others can be forced to open their commitments, the resulting list of random values has $2k = 2 * 256$ bits of entropy.

NIST [Barker and Kelsey 2012] states that the SHA family of hashes are suitable as randomness extractors. In [Barker and Kelsey 2012] it states that when using a hash function F in which $Y = F(S||A)$ then "If the input string S was assessed at $2n$ bits of min-entropy or more (i.e., $m \geq 2n$), then Y may be considered to have n bits of full entropy output". The value of A can be anything, including null, it is just additional data. This tells us that provided at least two mix servers provide good randomness values the output from the hashes will have $k = 256$ bits of entropy.

The usual subtlety arises if we consider the possibility that some authorities might use blocking to bias the output, *i.e.* might wait until learning the other authorities' random values and refuse to open their own commitments if they didn't like the result. (This could happen, for instance, in collusion with a corrupt printer.) This is why true coin-tossing protocols are more complicated than the simple one in this proposal. In practice, such a blocking authority would be removed quickly without having the opportunity to affect many bits of the output.

Clearly the same argument holds for the PRNG construction of Section 2.4.2, given appropriate assumptions about the PRNG.

6.3. Kleptographic attacks

The output of the printers is entirely determined by the randomness that is sent to them, and other publicly committed information given in Figures 1 and 2. Hence they have no opportunity to provide any information which may be skewed in a particular way. Correct information posted therefore cannot leak information from the printer. Incorrect information will be detected with some non-negligible probability by the ballot-generation audit processes.

Although the whole group of randomness generation authorities can collude to mount a kleptographic attack, a similar argument to that for vote privacy shows that a smaller collusion has insufficient information.

6.4. Selecting Ballots for Generation Audit

Clearly it is important that we use a suitable source of randomness for the selection of the ballots to audit. Ideally a publicly verifiable source would be good. This will require further investigation to see what procedures are possible in practice.

Of course, it is difficult to compute the appropriate amount of auditing for an IRV/STV election, especially in advance [Magrino et al. 2011; Cary 2011]. This question will have to be addressed for the project, but is out of scope for this paper. We expect that most of the IRV (*i.e.* single-seat) contests will have a relatively easy margin computation in practice. However, the full STV contests are a different matter and require significant further thought. One possibility is to announce the result, explain what quantity of cheating might have been possible given the amount of auditing that was done, and ask any election challenger to demonstrate a set of votes in which they win a seat and the number of changed votes is reasonably probable given the auditing.[15]

7. CONCLUSION AND FURTHER WORK

The application of Prêt à Voter to a real election has raised interesting research questions, including those that were considered solved under different assumptions. This paper presents a print-on-demand protocol with all the important desirable features of other protocols in the literature, but with much more feasible computational cost.

It would be interesting to try to extend these ideas to schemes such as Wombat or StarVote in which the ballot is directly encrypted by the voting device. A check for the proper use of randomness could easily be incorporated into the existing voter-initiated challenges these schemes offer. Indeed,

[15]Thanks to Ron Rivest for this suggestion.

the devices would not necessarily have to be online for this, as long as they were loaded with an authenticated list of proper random values to use. It's not surprising that this can be made to work since Markpledge 2 achieves the same result, albeit with a more complex voting protocol.

8. ACKNOWLEDGEMENTS

Many thanks to Craig Burton and Zhe (Joson) Xia for helpful comments.

REFERENCES

B. Adida. 2008. Helios: Web-based Open-Audit Voting. (2008). www.usenix.org/event/sec08/tech/full_papers/adida/adida.pdf and heliosvoting.org.

Ben Adida and C. Andrew Neff. 2009. Efficient Receipt-Free Ballot Casting Resistant to Covert Channels. In *EVT/WOTE 2009, Proceedings of the Electronic Voting Technology Workshop / Workshop on Trustworthey Elections, August 10th 2009, Montreal,Canada.* http://assets.adida.net/research/adida-neff-markpledge2.pdf

Michael Backes, Dario Fiore, and Esfandiar Mohammadi. 2013. Privacy-Preserving Accountable Computation. In *Proc. 18th European Symposium on Research in Computer Security*.

Elaine Barker and John Kelsey. 2012. *NIST DRAFT Special Publication 800-90B Recommendation for the Entropy Sources Used for Random Bit Generation*. Technical Report. NIST.

Josh Benaloh, Mike Byrne, Philip Kortum, Neal McBurnett, Olivier Pereira, Philip B. Stark, and Dan S. Wallach. 2011. STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. (2011). http://arxiv.org/abs/1211.1904

Dan Boneh, Ben Lynn, and Hovav Shacham. 2004. Short Signatures from the Weil Pairing. *J. Cryptology* 17, 4 (2004), 297–319.

Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. 2010. Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy. In *Proc. USENIX Security*.

David Cary. 2011. Estimating the Margin of Victory for Instant-Runoff Voting. In *USENIXAccurate Electronic Voting Technology WorkshopWorkshop on Trustworthy Elections*.

Gogolewski et al. 2006. Kleptographic Attacks on e-Election Schemes. In *International Conference on Emerging trends in Information and Communication Security*. http://www.nesc.ac.uk/talks/639/Day2/workshop-slides2.pdf.

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

Thomas R. Magrino, Ronald L. Rivest, Emily Shen, and David Wagner. 2011. Computing the Margin of Victory in IRV Elections. In *USENIXAccurate Electronic Voting Technology WorkshopWorkshop on Trustworthy Elections*.

Silvio Micali, Michael Rabin, and Salil Vadhan. 1999. Verifiable random functions. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE, 120–130.

Kim Ramchen and Vanessa Teague. 2010. Parallel Shuffling and its application to Prêt à Voter. In *Proc. USENIXAccurate Electronic Voting Technology Workshop*.

Alon Rosen, Amnon Ta-shma, Ben Riva, and Jonathan (Yoni) Ben-Nun. 2012. Wombat Voting System. (2012). http://www.wombat-voting.com.

P.Y.A. Ryan. 2007. *Prêt à Voter with Paillier Encryption, Revised and Extended*. Technical Report CS-TR-1014. University of Newcastle upon Tyne.

P.Y.A. Ryan and S. Schneider. 2006. Prêt à Voter with Re-encryption Mixes. In *European Symposium on Research in Computer Security (Lecture Notes in Computer Science)*. Springer-Verlag.

Daniel R. Sandler, Kyle Derr, and Dan S. Wallach. 2008. VoteBox: A tamper-evident, verifiable electronic voting system. In *Proc. 17th USENIX Security Symposium*.