*Cryptologia*, 34:236–257, 2010
Copyright © Taylor & Francis Group, LLC
ISSN: 0161-1194 print
DOI: 10.1080/01611190902894961

Taylor & Francis
Taylor & Francis Group

# Secure Electronic Voting—A Framework

STEFAN POPOVENIUC AND POORVI L. VORA

**Abstract** We describe a single framework in which to view the end-to-end-independently-verifiable (E2E) polling-place voting systems with a mixnet back-end. We use the framework to invent new systems that combine front and back-ends from existing systems.

**Keywords** back-end, end-to-end voting, front-end, mixnets, universal verifiability

## 1. Introduction

New voter-verifiable voting systems provide strong integrity and privacy guarantees [4, 7, 9, 11, 19, 20, 22]. In contrast to previous voting systems—where voters voted remotely using trusted computers—these voting systems are meant for use in polling booths, without access to trusted computational power at the time of voting. The security guarantees rely on the provision of paper receipts to voters. A receipt does not reveal the vote, but it can be used to ensure that votes are tallied correctly. The systems use different paper-based front-ends (for ballot preparation and the voting ceremony) and back-ends (for vote tallying and tally auditing); most of the existing literature on these systems describes a single, complete mechanism that does not typically address the components separately. This paper describes the existing *paper-based* E2E voting systems in a single framework, and by combining hitherto uncombined front and back-ends, presents new systems with hitherto unachieved properties. To keep the discussion focused, this paper presents *back-ends that use some form of mixnet*, and the corresponding front-ends. However, there is no reason why the front-ends presented cannot be combined with suitable back-ends that are not based on mixnets. For example, Scratch and Vote [3] uses a back-end based on homomorphic counters and can use either the PunchScan [21] or the Prêt à Voter [11] front-end.

The approach of the work is as follows. The paper-based voter-verifiable schemes use an innovative presentation of a two-part paper ballot—one part contains the encryption key, and the other the encrypted vote. The manner of ballot presentation enables a voter to check correct encryption through simple visual examination. The encryption is a symmetric-key encryption: either a stream cipher, or encryption with a simple substitution cipher. Inside the booth, the voter checks that the symmetric-key encryption is correct. The use of audits before, after, and/or during the election ensures that the keys are generated and printed correctly; the verifiable generation of keys is based on asymmetric-key or symmetric-key cryptography. Thus, the

Address correspondence to Poorvi L. Vora, Department of Computer Science, The George Washington University, Academic Center, 801 22nd St. NW, Washington, DC 20052, USA. E-mail: poorvi@gwu.edu

**Table 1.** Recent mixnet-based voter-verifiable voting systems and the components they use. ✓ stands for work that has already been done while ★ represents new concepts proposed in the current work

| | Onion mixnets | Punchscanian mixnet | Pointer mixnet |
|---|---|---|---|
| Visual Crypto | ✓ | ★ | |
| Pret a Voter | ✓ | ✓ | ★ |
| PunchScan | ★ | ✓ | ★ |
| Scantegrity II | ★ | ★ | ✓ |

front-end provides different ways to generate the key and encrypt the ballot, and the back-end to decrypt the ballots in a verifiable manner while preserving voter privacy.

The contribution of this work is two-fold: firstly, it is a survey of front-ends and back-ends of these systems. Secondly, it presents a single framework of which each of these systems is a special case, and it describes how front and back-ends can be combined (see Table 1) in ways that yield hitherto unachieved properties. The paper surveys three back-ends: mixnets using public keys and onions [14], punchscanian mixnets using precommitted paths and onions [21], pointer-based mixnets [10], with precommitted paths and no onions. Re-encryption mixnets are not studied as a separate case of mixnets; hence, for example, while the paper studies the original Prêt à Voter system [11], which uses encryption mixnets, it does not study in as much detail a variation which uses re-encryption mixnets [24]. The back-ends differ in the simplicity of the design, robustness and efficiency. Note that a mixnet-based back-end can be audited using a non-interactive zero-knowledge protocol (NIZKP) or a more efficient but less private protocol such as a randomized partial audit [16]. This paper briefly describes both types of audits, though the systems surveyed have, for reasons of efficiency, typically used a variation of the randomized partial audit. We also survey four front-ends: visual cryptography [7], shuffling the order of the candidates (Prêt à Voter [11]), indirection-based encryption (PunchScan [21]), and a front-end that is an overlay to the current optical scan (Scantegrity II [9]). In order to maintain its focus on the use of mixnet-based back-ends, and on the use of paper ballots as a means of vote encryption, this paper does not cover in as much detail several excellent voting systems such as the remote voting system (Helios [2], or Scratch & Vote [3]), which uses a back-end based on homomorphic counters. Proofs are outside the scope of this paper.

The paper is organized as follows. Section 2 presents related work. Section 3 presents a front-end model that generalizes the front-ends of the various existing example systems; Section 4 demonstrates how the front-ends of existing voting systems are specific instances of the general front-end. Section 5 demonstrates how existing back-ends are specific instances of a generalized back end. Section 6 presents new systems obtained by combining front and back-ends, and Section 7 presents conclusions. Table 1 presents the existing systems and the contributions of this paper.

An early draft of this paper appeared in *WOTE 2008* [22].

## 2. Related Work

In Scratch & Vote [3], Adida and Rivest explain how their homomorphic scheme would work with two types of front-ends (Prêt à Voter and PunchScan). Van de

Graaf [25] notices similarities between PunchScan and Prêt à Voter and describes a detailed mechanism that allows the use of a Prêt à Voter front-end with a PunchScan back-end. Lundin [18] provides a framework for all the aspects of voting—including registration, election method, election mechanics, election management, and transfer methods—and decomposes the voting system in its totality into components that can be interchanged. Chaum [5] suggests writing a common XML format for representing the results of the scanning of PunchScan and Prêt à Voter ballots to allow the interchange of scanning technologies.

Our work looks at a larger numbers of systems and is more general than that of Van de Graaf [26] and Scratch and Vote [3], looking at a larger number of systems. We also present a clear distinction between front and back-ends, and suggest general ways of interconnecting them to obtain voting systems with properties distinct from voting systems previously presented. At the same time, our work is more focused than that of Lundin [18]. We look only at two of the components that make an election process work from start to finish, and providing details about each possible combination of front and back-ends.

There is an entire body of work on the use of verifiable mixnets for the purposes of voting. Ben Adida's doctoral dissertation [1] provides an excellent recent survey. Related to the classical mixnet back-end described in Section 5.1, is the re-encryption mixnet of Park et al. [13], where the random value for each mixnet server is algebraically combined with the existing numerical values, and not concatenated; this leads to onions that do not increase in size with the number of mixes used. Key contributions in the area of mixnet verification include the *universally verifiable mixnet* of Sako and Kilian [25], which was the first to provide a proof of correctness for the mixnet output that could be verified by anyone, including non-voting observers. Later work provides more efficient proofs that exploit the underlying algebraic structures of the particular cryptosystems; see, for example, the proofs of Furukawa and Sako [16], and Neff [20]. The *randomized partial audit* of Jakobsson et al. [17], provides a more efficient correctness proof that is, however, not zero-knowledge. The two types of audits are described in more detail in Section 5.1.

## 3. The General Front-End

The front-end represents the manner in which the ballots are presented to the voters and the voters interaction with the system. In the voting protocol, the front-end represents a voter-verifiable encryption of the vote; the receipt bears the encrypted vote. The ballots presented in this work fall in two categories. The first category is that of symmetric ballots: those that have two parts, each of which is sufficient to recover the vote (the original scheme using visual cryptography [12, 27] described in Section 4.1, and PunchScan described in Section 4.2). On visual examination of the two parts appropriately laid out, the voter is able to confirm that each part bears the encryption of her vote. The second category is that of asymmetric ballots: those that have two parts, of which a specific one is needed to recover the vote (Prêt à Voter described in Section 4.3 and Scantegrity II described in Section 4.4). In this category too, the voter visually examines both parts of the ballot laid out in a particular manner (side by side) to confirm that the encrypted vote represents her ballot. Scantegrity II requires an indirection not present in other schemes, we discuss this in more detail later. We define symmetric and asymmetric ballots more formally below.

**Definition 1 (Symmetric ballot).** A symmetric ballot is a ballot that has n parts and

- when combining the n parts the clear text vote is available;
- a single part or a combination of any number of parts strictly less than n does not reveal anything about how a voter voted; and
- any single part can become the receipt.

**Definition 2 (Asymmetric ballot).** An asymmetric ballot is a ballot that has n parts and

- when combining the n parts the clear text vote is available;
- a single part or a combination of any number of parts strictly less than n does not reveal anything about how a voter voted; and
- a designated part always becomes the receipt.

### 3.1. The General Receipt

We now describe more formally a receipt. In all the voting systems we study in this paper, the voter gets a receipt of serial number $s \in \mathcal{S}$, where $\mathcal{S}$ is the set of all serial numbers. A vote is $v \in \mathcal{V}$, where $\mathcal{V}$ is the set of all possible votes. $\mathcal{E}$ is the encryption cipher used to generate the receipt, $\mathcal{E} = (\mathcal{V}, \mathcal{R}, \mathcal{K}, E, D)$, where $\mathcal{R}$ is the space of all ciphertexts for $\mathcal{E}$; $\mathcal{K}$ the keyspace, and $E$: $\mathcal{K} \times \mathcal{V} \rightarrow \mathcal{R}$ and $D: \mathcal{K} \times \mathcal{R} \rightarrow \mathcal{V}$ the encryption and decryption functions respectively. Finally, a function $f$: $\mathcal{S} \rightarrow \mathcal{K}$ provides the association between the key and the receipt. Note here that the encryption schemes used are deterministic; serial numbers are unique, keys are rarely, if ever, reused, and the encryption cipher is a symmetric key encryption. The private function $f$ may itself be a keyed function, with a keyspace that is distinct from $\mathcal{K}$. Additionally, the inverse of $f$ may be a public function; however, neither condition is necessary—in particular, $f$ need not be bijective. It is important, however, that, given $k'$ and $s$, it can be checked that $k' = f(s)$ with high probability. Using the previous notation, the receipt obtained by a voter in a correct instance of the voting protocol is the triplet ($s$; $x$; $E(f(s), v)$), where $v$ is the vote; and $x$ represents any other information, such as onions and commitments made by the voting system. Note that $E(f(s), v)$) denotes the encryption of vote $v$ using the encryption key $f(s)$ associated with serial number $s$. This receipt is the only vote-related information to enter the back-end of the voting system, and the tally is constructed from this information.

**Definition 3 (Receipt).** A receipt is the triplet ($s$, $x$, $r$), where s is the serial number, x any additional information on the receipt, and r the purported encrypted vote.

Note that we are not always assured that $r = E(f(s), v)$ and that $x$ is correctly formed.

### 3.2. The General Printing Audit

All front-ends discussed here are based on paper ballots; this enables the voter to retain a physical artefact of the voting process (a paper receipt), and provides some resemblance to voting with paper ballots, a process familiar to most voters. (We do not claim that all the E2E voter-verifiable voting systems are as easy to use as regular paper-ballot voting, we simply indicate that the choice of a paper receipt, as opposed to any other type of receipt, has not been arbitrary by the inventors of the protocols). At some point, hence, the ballots and/or receipts need to be printed. Verifying the correctness of the printed ballots is referred to as a printing audit, and is carried out either before, during, or after the election.

A correct printing provides a *Correct Encryption*, which enables the system to correctly decrypt the vote. There are two possibilities for the mixnet-based decryption.

1. The key may be referenced through $s$—either through a lookup table with entries individually committed to and indexed by $s$ (these commitments are opened at the time of the audit); or by the use of similar commitments to keys for the keyed function $f$. In this case, a printing audit checks that the key used for encryption is indeed $f(s)$.
2. The receipt may bear additional information in the form of an onion that is used by successive mixes to effectively decrypt the receipt using the correct key. In this case, the printing audit checks that the key represented by the onion(s) is that used for encrypting the vote.

**Definition 4 (Printing Audit)**. A printing audit is a process that ensures each voter, with high probability, that the entity that produced the ballot did so correctly (consistent with the other published data about the election).

The most general printing audit allows each voter to choose a number of ballots, one to vote with, and the others to spoil and verify. Because the choice of which ballots to spoil is made at random by the voters, the probability that an incorrect printing is undetected drops exponentially as the number of misprinted ballots increases. Variations on this method are possible, for example the voter does not actively spoil a ballot, as it becomes a natural result of the voting process: some voters simply make mistakes when filling them in, spoil them, and ask for a second ballot; the spoilt ones may be checked by the auditors. Unfilled ballots may be checked at the end of an election. This type of audit works for both symmetric and asymmetric ballots.

The printing of symmetric ballots may also be audited by examining, for correctness, the ballot half chosen for a receipt. Because the system cannot predict beforehand which half will be chosen, the probability that an incorrect printing is undetected decreases exponentially with the number of incorrect ballots.

### 3.3. *Other Proofs of Correct Encryption*

Several of the voter-verifiable systems do not perform vote encryption through the use of paper ballots. While we do not focus on these in this paper, in this section we describe briefly how a voter may check for correct encryption in these techniques. Consider, for example, the front-end of the Helios voting system [2], based on Benaloh's Simple Verifiable Voting protocol [4]. In this voting system, an encrypted vote is generated by a ballot preparation system. A voter, or anyone, can generate as many encrypted votes as he or she wishes, and spoil several in order to audit the encryption. If the ballot preparation system cannot predict which encrypted votes are audited, it will be caught with high probability if it cheats on a few encryptions. The voter casts an unspoilt encrypted vote at a different, ballot casting, machine. As should be clear, these audits are very similar to the printing audits described above.

### 3.4. *Types of Receipts*

After all ballots are cast, the voting system makes available all receipts on a public bulletin board. The voter can check that the triplet (*s, x, r*) is among these. If she notices a discrepancy, she can file a complaint. There are two types of receipts the voter can get: proof receipts and indication receipts.

**Definition 5 (Proof Receipt)**. A proof receipt is a receipt held by the voter that is non-repudiable by the voting system.

A proof receipt is one that is produced and "signed" by the voting system itself (or by an election official). If the signature is a digital signature, the voter should be able to check it for correctness at a time when he or she can challenge the signature. On the other hand, the signature could correspond to a physical seal or stamp, which a voter may more easily verify without recourse to trusted computation. When a voter holds a proof receipt that is inconsistent with the information on the public bulletin board, and the digital signature scheme used is assumed secure, it is sufficient evidence that the bulletin board contains erroneous information. This, in turn, is an irrefutable indication that something went wrong with the election.

**Definition 6 (Indication Receipt)**. An indication receipt is a receipt held by the voter and that, in case the voter notices a discrepancy between the proof receipt and the public data, can be used to trigger additional investigation.

Indication receipts provide only a hint that something might have gone wrong, but additional evidence needs to be provided in order to prove that something went wrong. Indication receipts can be produced by voters themselves and are not "signed" by any election authority; these receipts are hence not non-repudiable.

We now describe in more detail the "voting ceremony" for several front-ends, and indicate how the front-ends are specific instances of the general description above. The voting ceremony consists of the specific steps a voter needs to take to cast a ballot, and, eventually, to verify later that the ballot was printed correctly and recorded as cast. We will describe the receipt; in particular, we will describe the encryption process. We will intentionally not describe the value in $x$, as that depends on the back-end.

## 4. Example Front-Ends

In this section, we describe the existing front-ends of four paper-based voter-verifiable voting systems with mixnet-based back-ends, and demonstrate how they correspond to specific instances of the general front-end of Section 3.

### 4.1. *Ballots using Visual Cryptography*

Chaum [6] describes a ballot made of multiple parts, such that the combination of all parts makes the text readable; but no information is revealed about the vote when only a subset of the parts is available. The layer used as an encryption of the vote can later be decrypted by a mixnet to recover the vote unambiguously. The first instantiation of this idea used visual cryptography [19].

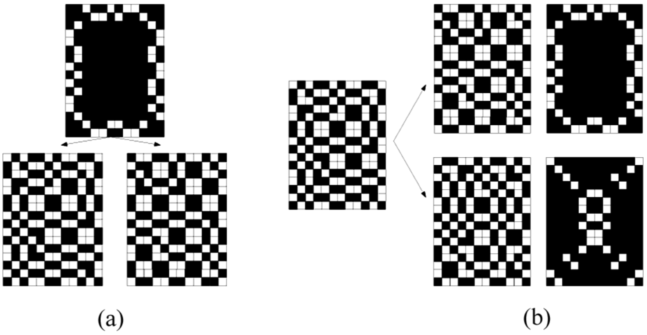

**Figure 1.** A sample ballot using visual cryptography. (a) A ballot containing a vote for ZERO. (b) One layer can represent with equal probability a vote for either ZERO or X, depending on what the other layer is.

A detailed explanation of how the layers are built may be found in [12, 27]. On the top layer, the odd pixels are generated pseudo-randomly, while on the bottom layer, the even pixels are generated pseudo-randomly. The rest of the pixels are generated in a way that constructs the clear text image of the ballot only when the two layers are overlaid. The voter is able to read her vote when the two pages are overlaid (Figure 1a), but when looking at a single layer, no information is leaked about the voter's choice (Figure 1b).

A more formal description is as follows. A correct receipt is of the form $(s, x, v \oplus k_a)$ where $\oplus$ denotes bitwise XOR, $k_a$ represents the key for the layer corresponding to the receipt. The other layer is encrypted using key $k_{\bar{a}}$. The keys are generated using different seeds for a pseudo-random number generator; $k_a = f_a(s) = F(Sign(s,\ p_a))$, where $Sign(s,\ p_a)$ is the deterministic digital signature of the serial number using the private key of the polling machine that corresponds to layer $a$, and $F$ the public pseudo-random number generator. If the two receipts are $(s_a,\ x_a,\ r_a)$ and $(s_{\bar{a}},\ x_{\bar{a}},\ r_{\bar{a}})$, because of the manner in which the bits in each layer are presented, the voter can visually check that $r_a \oplus r_{\bar{a}} = v,\ s_a = s_{\bar{a}}$. The information in $x$ depends on the back-end used.

### *The Voting Ceremony*

On election day, the voter goes to her assigned polling place, authenticates herself as a legitimate voter and uses a touch screen to make the desired selections. When finished, the computer prints the two layers, the voter checks that, when the two layers are overlaid, her vote is shown. The voter chooses one of the layers as a receipt and watches the other one being destroyed. The computer prints additional information on the receipt, that allows anyone to check that the pseudo-random pixels on the chosen layers have been constructed correctly. A digital signature is also printed. After election day, any voter can go to the election authority web site, enter the serial number for her ballot, check that the ballot is there and that it matches the page she possesses: the pixelized image and the strings on the receipt should be the same as the ones posted on the web site. This voting scheme hence provides a proof receipt if voters can check digital signatures inside the polling booth, and uses a symmetric ballot. Correct printing is checked by auditing the ballot half that forms the proof receipt.

*Advantages and Disadvantages*
The advantages of this approach are: the high degree of generality (it can accommodate any type of contest, including write-ins), that the receipt is created automatically, that it allows a fixed order of candidates, that it offers excellent privacy (except for the fact that the voting machine knows the clear text votes), and that there is no need for a strict chain of custody of the ballots. The disadvantages are: the voters are not familiar with the receipt interface, the order of events must be precise, the alignment of pixels is difficult, the receipt is difficult to check by the voter, it is very difficult to implement in practice (one reason is the alignment of the two pages), it does not accommodate disabled voters, it does not allow manual recounts, the cost is very high (because one machine per booth is needed), and the administration of the system is difficult.

### 4.2. Ballot with Indirection

To allow the same separation of information as in the previous case, the following technique can be used: on one page each candidate is associated with a random symbol; on another page the same set of symbols appears in a random order. For convenience the two pages can be overlayed, with the top page having holes and the symbols on the bottom page being visible through the holes. (Figure 2a). This technique was first proposed in PunchScan [8] and therefore this style of ballot is known as a PunchScan ballot.

In PunchScan, the voter uses a dauber to mark the selection of candidates. The diameter of the ink disc is greater than the diameter of the hole punched through the top page, therefore the dauber leaves a mark on both the top and the bottom ballot pages. Figure 2b contains a ballot voted for "Yes." Because the order of the symbols on the two pages of a ballot is different (and independently and uniformly distributed), one cannot determine which mark is for which candidate by viewing only one voted page. The association of candidates with symbols, and the order of the symbols on the bottom page, can be uniformly random, or pseudorandom.

Thus, in PunchScan, the receipt is of the form: $(s, x, E(f(s), v))$ where $E$ is viewed as a permutation of the plaintext space (all encryptions are trivially permutations of plaintext space) composed of two distinct permutations: the first is the association of candidate choice with dummy variables (viewed as a map of candidates in canonical

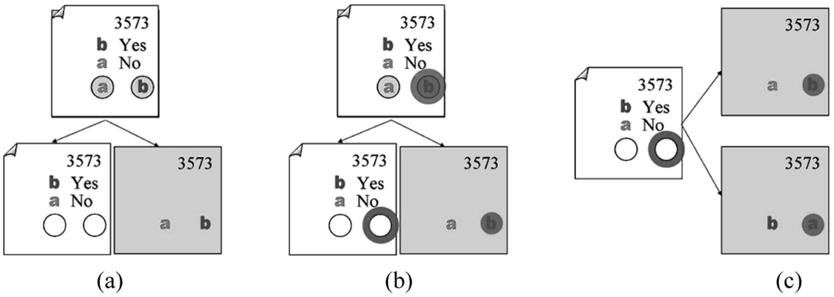

**Figure 2.** PunchScan's ballot. (a) A sample Punch-Scan ballot. When the two pages are over-layed, the symbols on the bottom page are visible through the holes. (b) A voted ballot. Looking at each layer individually, one cannot say that the mark is for "Yes" or for "No." (c) Given only one layer of the ballot, the marks on that layer are equally likely to represent a vote for any candidate.

order, such as alphabetical order, to dummy variables in canonical order) and the second the association of dummy variables with positions (again the map can be between canonical orderings). With abuse of notation, using the same notation for the key and the encryption function it represents, the key $f(s) = \pi_a(s) \circ \pi_{\bar{a}}(s)$ is the composition of two permutations, $\pi_a$ and $\pi_{\bar{a}}(s)$, one for one of the layers $a$ and the other for the other layer $\bar{a}$, each a well-defined function of $s$.

*The Voting Ceremony*

On election day, the voter goes to her assigned polling place, authenticates herself as a legitimate voter, and before seeing the ballot, commits to which page to keep as a receipt. In the privacy of a booth, the voter marks the hole that contains the symbol associated with her favorite candidate, and, when done, scans the page chosen in the first step, keeps it and shreds the other one. After election day, any voter can go to the election authority web site, enter the serial number for her ballot, check that the ballot is there and that it accurately resembles the page she possesses: her marks are recorded correctly and the order of the symbols on her receipt is the same as the order posted. The receipt is signed, and this method provides a proof receipt using a symmetric ballot if voters can check digital signatures inside the polling booth. Correct printing is ensured through the auditing of the receipts.

*Advantages and Disadvantages*

The advantages of this method are: the receipt is created automatically and is easily checkable by the voter, it allows a fixed order of the candidates on the ballot, it offers excellent privacy (the scanner does not know the clear text votes, however the printer does know all clear text votes), the cost is low, the dispute resolution is easy, and it accommodates disabled voters (see the PunchScan website for a brief description of such capability, which also follows for Prêt à Voter; this capability is also described in more detail in [13]). The disadvantages are: it does not accommodate write-ins (but it accommodates most types of contests), the voters are not familiar with the voting interface (the indirection may cause usability problems), its privacy properties require a strict chain of custody before the ballots reach the voters, and it does not allow for a manual recount.

### 4.3. Permuting Candidate Order

Prêt à Voter [11] proposes a simplification of the two-part visual cryptography ballot presented in Section 4.1, see Figure 3a. The ballot is printed on a single page of normal paper, with the names of the candidates on the left and the places to mark on the right. A voter makes a mark next to her favorite candidate (Figure 3b). The names of the candidates are permuted on each ballot and when the left part is separated from the right part, the marks on the right are no longer associated with candidates (Figure 3c). This ballot style is an example of a ballot that has two parts (left and right) but the information is distributed asymmetrically in the two parts. Thus the Prêt à Voter receipt is of the form ($s$, $x$, $E(f(s), v)$) where $f$(s) is a permutation.

*The Voting Ceremony*

On election day, the voter goes to her assigned polling place, authenticates herself as a legitimate voter, gets two ballots from the election officials and chooses one to audit (for the printing correctness check) and one to use for voting. In the privacy

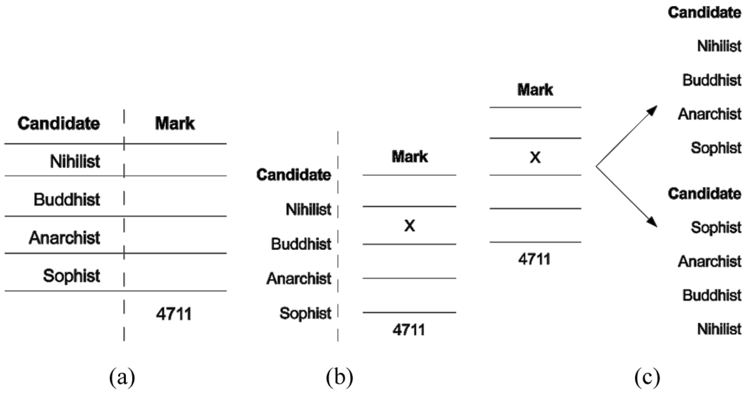

**Figure 3.** Prêt à Voter ballot. (a) A sample Prêt à Voter ballot. A permuted list of candidates is on the left. (b) A voted Prêt à Voter ballot. When the right side is separated from the left side the mark is not a clear vote anymore. (c) Given only the right side, the mark is equally likely to represent a vote for any candidate.

of the voting booth, the voter makes an X on the right side of the ballot, next to her favorite candidate. The voter separates the list of candidates (on the left) from the marks (on the right), destroys the left side and scans the right side. The marks are recorded and made public. The scanned side is kept by the voter and anytime after election day, the voter can go to the election authority web site, enter the serial number for her ballot, check that the ballot is there and that it accurately resembles the page she possesses. The receipt is signed, and this method provides a proof receipt using an asymmetric ballot if voters can check digital signatures inside the polling booth. Correct printing is ensured through audits of spoiled ballots.

*Advantages and Disadvantages*

The advantages of this method are: the voters are somewhat familiar with the interface,the receipt is created automatically and is easily checkable by the voters, it accommodates disabled voters, and it offers excellent privacy (the scanner does not know the clear text votes) at a low cost. The disadvantages of the method are: it does not accommodate write-ins (but it accommodates most types of contests); it does not allow a fixed order of candidates; its privacy properties require a strict chain of custody before the ballots reach the voters; and it does not allow for a manual recount.

### 4.4. *Standard Optical Scan Ballot, Encoded Receipt*

Scantegrity [10] and Scantegrity II [9] address the usability concerns of Punch-Scan while keeping the order of candidates fixed on all ballots. A Scantegrity ballot contains two asymmetrical parts, but because the two parts are never separated, it is printed on a normal piece of paper that will not be divided in any way. One part of the ballot is a normal optical scan ballot, which can be scanned and used by any certified optical scan voting system. The other part is a set of confirmation numbers associated with the candidates (e.g., printed next to the candidates). The association is different on each ballot. See Figure 4 for a sample Scantegrity II ballot.

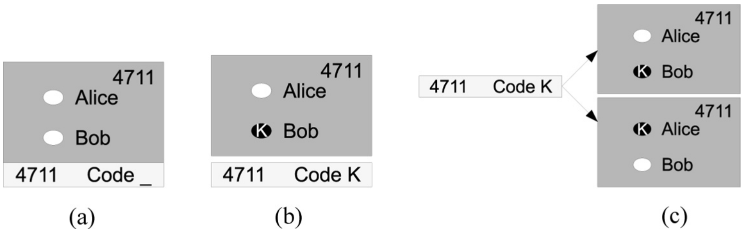

**Figure 4.** Scantegrity Ballot: Blue is the ballot form, and Yellow is the receipt. Typically, the indication receipt may contain the serial number and the confirmation number. (a) A sample Scantegrity ballot, just like a regular optical scan ballot. (b) A voted Scantegrity ballot. When the oval is filled in, a confirmation number appears. (c) The symbols on the receipt may correspond to any candidate.

The difference between Scantegrity and Scantegrity II is that the voter only gets the confirmation numbers for the candidates she is choosing in Scantegrity II, while in Scantegrity the voter is able to see the confirmation numbers for all the candidates. The immediate benefit is in the dispute resolution process: voters that claim to have their ballots registered improperly must provide the confirmation numbers, which are random and hard to predict. The election authority can then discard the complaints that contain confirmation numbers that do not appear on the indicated ballot and race. In Scantegrity the receipt is of the form ($s$, $x$, $E(f(s), v)$) where $f(s)$ is a permutation of some standardly ordered symbols (alphabetically ordered letters), whereas in Scantegrity II $f(s)$ is a mapping between candidates and randomly chosen codes.

### *The Voting Ceremony*

On election day, the voter goes to her assigned polling place, authenticates herself as a legitimate voter, gets two ballots from the election officials and chooses one to audit (for the printing correctness check) and one to use for voting. In the privacy of the voting booth, the voter marks the ballot as a normal optical scan ballot. On a separate piece of paper, she writes down the confirmation numbers associated with the voted candidates, tears off the bottom part of the ballot, (which also contains the serial number of the ballot) and keeps it. The ballot is scanned by a regular optical scanner. After election day, any voter can go to the election authority web site, enter the serial number and check that the symbols she wrote down are on the web site.

Because the receipt the voter gets is an indication receipt (as opposed to a proof receipt), if the voter sees on the web site a different set of symbols then the ones on her own piece of paper, she has to have a way of challenging the records on the bulletin board. Depending on the length and unpredictability of the confirmation numbers, a set of dispute resolution techniques are possible; see [9] and [10] for details. This method hence provides an indication receipt using an asymmetric ballot. Correct printing is ensured through the auditing of spoiled and unused ballots.

### *Advantages and Disadvantages*

The advantages of Scantegrity II are: the voters are highly familiar with the interface; it is highly usable by the election officials, the voters can easily check their receipt, the dispute resolution process is easy; it allows for a fixed order of the candidates, it accommodates disabled voters; it allows a manual recount, the cost is very low; and it is very easy

**Table 2.** Evaluation of various types of ballots

| | Visual crypto | PunchScan | Prêt à Voter | Scantegrity II |
|---|---|---|---|---|
| Generality | Any type of contest | Most practical contests | Most practical contests | Most practical contests |
| Familiarity with the interface | Low | Low | Medium | High |
| Receipt creation | Automatic | Automatic | Automatic | Requires voter effort |
| Voter verification | Difficult | Easy | Easy | Easy |
| Supports write-ins | Yes | No | No | No |
| Fixed order of the candidates | Yes | Yes | No | Yes |
| Ease of implementation in practice | Difficult | Easy | Easy | Easy |
| Accommodates disabled voters | No | Yes | Yes | Yes |
| The voting machine knows the clear text votes and can compute the tally | Yes | No | No | Yes |
| Chain of custody to protect privacy | No | Before the ballot reaches the voter | Before the ballot reaches the voter | After the ballot is marked |
| Manual recount | No | No | No | Yes |
| Cost | High | Low | Low | Low |
| Ease of administration | Difficult | Moderate | Moderate | Easy |
| Ease of dispute resolution | Easy | Easy | Easy | Easy |

to administer. Another advantage is that the voting machine can compute an independent tally; while this is also true of the visual cryptography based system, it is more important in this instance because it allows the voter-verifiability to be an unobtrusive aspect of a regular voting system based on optical scan. In such a system, it is possible to compute vote tallies at the precinct or scanner levels. The disadvantages of the method are: it does not accommodate write-ins, the voting machine knows the clear text votes, and it needs a strict chain of custody after the voters mark their ballots to protect privacy.

Table 2 summarizes the advantages and disadvantages of the four types of front-ends.

## 5. Example Back-Ends

The back-end is responsible for producing clear text ballots from the encrypted receipts produced during the voting ceremony. The process is typically fully auditable by anyone, yielding universal verifiability, while preserving the secrecy of the votes. While the verifiability of the election depends on voters verifying their receipts, and is hence not completely universally verifiable, it is possible to have a back-end that is universally verifiable. In a general sense, in mixnet-based back-ends this works as follows: in one approach the information required for decryption is composed of two or more parts, either stored in some form of database, as in PunchScan, or printed on the ballot in the form of an onion, as with classical mixnets. Ballots are shuffled after the use of each piece of information; hence the encrypted ballot has a path through the mixnet, based on its position in each shuffle. When the decryption information is stored on the ballot the path is chosen on the fly; when it is not, the path is pre-determined and the corresponding information in the database is linked by pointers. In another approach, as with pointer-based mixnets, in Scantegrity, the shuffle (a permutation of ballot order) is combined with decryption (a permutation of candidate order).

In two of the three cases presented here, the back-end is also responsible for initially creating the blank ballots.

Three main techniques are presented:

- classical mixnets using public keys and onions. The path followed by a vote is determined on-the-fly.
- punchscanian mixnet using pre-established and committed paths and onions.
- pointer-based mixnet with pre-established paths and no onions.

We briefly describe each back-end and suggest simple ways to connect hitherto uncombined back-ends with front-ends in Table 1.

### *5.1. Traditional Mixnets*

Mixnets have been classically associated with onion routing because the payload can be viewed as an onion, with multiple layers of encryption; each mix strips off one of the layers. Besides the onion, the payload also contains a value (a ballot in the case of voting systems). After removing one layer of encryption from the onion, a mix finds a seed (sometimes called a germ) that is used to transform the value in the payload. This way, the output value is uncorrelated with the input value.

We now briefly explain how onion routing works, referring to, for example, [12]. In general, the payload is a pair (`Onion, Ballot`). Thus, when the back-end is a traditional mixnet, the value of $x$ for all the front-ends contains the onion. The serial number from the receipt is stripped after voters have checked the presence of the receipt on the bulletin board, and the triplet ($s$, $x$, $r$) is reduced to the pair ($x$, $r$), referred to as (`Onion, Ballot`). (`Ballot` hence represents the encrypted vote). For a particular mix $j$ and a particular input-output pair, the input is $\texttt{Payload}_j = (\texttt{Onion}_j, \texttt{Ballot}_j)$ and the output is $\texttt{Payload}_{j+1} = (\texttt{Onion}_{j+1}, \texttt{Ballot}_{j+1})$. The relation between the two onions is

$$\texttt{Onion}_j = Enc_{(\texttt{PublicKeyOfMix}_j)}(seed_j, \texttt{Onion}_{j+1}) \tag{1}$$

where $Enc_{(\texttt{PublicKeyOfMix}_j)}$ represents encryption with the public key of mix $j$ and the comma represents concatenation. The $(j+1)$th onion is obtained by decrypting the $j$th onion and removing $seed_j$: $\texttt{Onion}_j = Dec_{(\texttt{PrivateKeyOfMix}_j)}(\texttt{Onion}_{j+1}) \setminus seed_j$, where \ denotes removal from a string. The size of the onion grows linearly with the number of layers. One manner in which the relation between the input and the output ballot may be viewed is [23]:

$$\texttt{Ballot}_{j+1} = F_j(seed_j)(\texttt{Ballot}_j) \tag{2}$$

where $F_j(seed_j) \in G$, $\forall j$ for group $G$ with operation $\odot$, and $F_j$ is a public function. An important aspect is that the Onion and the Ballot have to travel together through the mix. Thus $\bigodot_j F_j(seed_j)(\texttt{Ballot}_j) = D(f(s), \texttt{Ballot}_j)$ decrypts the encrypted receipt. Note that $seed_j$ is a function of $s$.

The traditional mixnet is used as the back-end of the voting scheme proposed by Chaum that uses a front-end based on visual cryptography (see Section 4.1), and by Pret a Voter (see Section 4.3). In the scheme of Chaum, $G$ is the set of all bitwise XORs acting on $n$-bit strings, and $F_j(seed_j)$ corresponds to a bitwise XOR using the pseudo-random string generated using $seed_j$. The composition of the processing of individual mixnet entities corresponds to the bitwise XOR of the receipt bitstring with the bitstring used to encrypt it. In Prêt à Voter, $G$ is the set of permutations on sets of size $c$, the number of candidates, $\odot$ is permutation-composition and $F = \mathrm{P} \circ h$ where $h$ is a one way function and $P$ is a function that generates a permutation on a set of size $c$ given a seed. The composition of the processing of individual mixnet entities corresponds to the inverse of the permutation used for encryption. If a mixnet processes a single rance, computing the permutation that a single mix applies requires $N \times \log(N)$ operations (e.g., for sorting the output), $N$ operations for removing a layer of the onion and another $N$ operations for computing the outputs. This needs to be multiplied by the number of mixes and again by the number or races.

A mixnet may be audited by either providing a zero-knowledge proof of correctness or using a randomized partial checking (RPC) technique [17]. We describe these next.

### *Mixnet Audits: Randomized Partial Checks*

In a randomized partial audit, the mix is required to reveal $seed_j$ for a significant fraction of its inputs (or outputs). Having the seed, the auditor (sometimes called the challenger or verifier) can check Eqs. (1) and (2) for all the revealed input-output pairs. The inputs or outputs revealed in the different mixes are chosen so as to protect vote privacy. For example, in one variation, the first mix reveals the input-output pairs corresponding to a randomly-chosen half of the ballots. The second mix reveals the input-output pairs corresponding to the other half. Consequent mixes alternately reveal input-output pairs corresponding to half of the ballots, so that the input-output pair of a single ballot is never opened by two or more consecutive mixes. This ensures that a plaintext vote is not traceable to the corresponding encrypted vote. This audit does, however, reduce the number of possible encrypted votes a single plaintext vote can correspond to. In another variation, each mix opens input-output pairs corresponding to half of the ballots, chosen uniformly at random. The probability that the link between a single output plaintext vote and its corresponding input encrypted vote will be revealed decreases exponentially with the number of mixes. Several voting systems use a variation of randomized partial checking, including the voting system of Chaum based on visual cryptography, Prêt à Voter, PunchScan, and Scantegrity.

*Mixnet Audits: Zero-Knowledge Proofs*

Unlike randomized partial checks, zero-knowledge proofs of mixnet correctness do not reveal any additional information about the relationships between encrypted and plaintext votes than is already available—through the tally, and through other information external to the voting system, such as demographic information, and information obtained from exit and opinion polls. The first of these proofs is due to Sako and Kilian [25], who use a cut-and-choose protocol. Benaloh describes a typical general zero-knowledge proof of mix correctness [4] as follows. A set of encrypted ballots, $\mathcal{B}$, may be verifiably shuffled to produce a ballot set $\mathcal{B}'$ as follows. The prover makes several sets of ballots by re-encrypting exactly once each ballot from $\mathcal{B}$, and shuffling all of them; these sets of re-encrypted and shuffled ballots are denoted $\mathcal{B}', \mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$. For each $\mathcal{B}_i$, the prover is asked to show one the following, chosen at random: (a) that $\mathcal{B}_i$ is obtained by re-encrypting and shuffling the ballots from $\mathcal{B}$, or (b) $\mathcal{B}_i$ is obtained by re-encrypting and shuffling the ballots from $\mathcal{B}'$. If ballot set $\mathcal{B}'$ is not an encrypted and shuffled version of ballot set $\mathcal{B}$, the verifier will not be able to respond correctly to all challenges. The Web-based voting system Helios [2] uses a non-interactive variant of a zero-knowledge proof. Neff has proposed a more efficient proof, (which exploits the structure of the ElGamal encryption scheme used for the mixnet) as well a voting system based on it [20].

*Advantages and Disadvantages*

The advantages of onion mixnets are their truly distributed nature, support for dynamic paths and the possibility of setting up the system before the details of the elections are known. The disadvantage is low efficiency, both when processing the ballots and during the audit process.

### 5.2. Punchscanian Mixnet

A punchscanian mixnet [21] has been viewed as an integral part of PunchScan itself, however we describe how it may be used with other front-ends, after first providing a brief overview. The path through a punchscanian mixnet is fixed a-priori and commitments to it are published a priori; the paths and permutations are pseudorandomly generated. The advantage of having pre-set paths is that the onions do not have to be part of the payload anymore. The notion of an onion gets degraded to a chaining of secret seeds, which are fixed along the path. The payload becomes only the ballot itself, which carries the encrypted selection of the voter. In a punchscanian mixnet $\texttt{Payload}_j = (\texttt{Ballot}_j)$ and there is no relation between the degraded onions; that is, there is no variable $x$ in the receipt. Because the paths are pre-committed to, and only the mixnet knows the seeds, the mixnet itself produces the ballot (as opposed to the voting machine producing the onions). In this setting, the mixnet is a single entity, and not composed of several entities; however, this single entity may be split among several using standard secret sharing approaches. If RPC is used to audit the mixing, the single entity consists of two lists for the purposes of the audit and hence:

$$\texttt{Ballot}_j = F(seed_{j1}) \odot F(seed_{j2})(\texttt{Ballot}_{j-1}), \tag{3}$$

where $F$ is a public function. Having access to only one of the three elements in the equation does not leak any information about the other two. The commitments to the seeds can be independent, or can be blended into the commitments to the paths. While traveling through the mixnet, the ballot is transformed according to Eq. (2).

After the ballots are produced, they are publicly committed to. To ensure that the produced ballots are consistent with the seeds used to generate them, a significant fraction of the ballots are randomly chosen to be opened, and Eq. (3) is checked for all of them. If the checks are successful, the ballots that were not opened are also consistent, with high probability, with the paths and the seeds actually used for the decryption. The ballots that are opened need not be printed on paper. To ensure that the ballots that survived the audit are printed correctly, another audit (called a printing audit) has to be performed. If all the ballots are initially printed, then the printing audit can be combined with the mixnet correctness audit.

If a punchscanian mixnet processes a single rance, computing the two permutations requires $2 \times N \times \log(N)$ operations (e.g., assuming sorting is used to generate the permutations), and another $2 \times N$ operations for computing the outputs. These operations are for the entire mixnet (as opposed to each mix) and need to be multiplied by the number of races.

*Advantages and Disadvantages*

The major advantage is the high efficiency. Millions of ballots can be tallied in minutes.[1] The disadvantages are the central nature of the authority, and the need to know the details of the election before setting it up.

### 5.3. Pointer-based Mixnets, or Mixnets with No Explicit Group Operation

In its traditional form, the payload of the mix consisted of an onion and a ballot. A first simplification step, as in the punchscanian mixnet, was to separate the two, absorb the onions into the mixnet and require only the ballot to travel. A second step is to remove the onion altogether. The onion does not vanish from a conceptual perspective, but is absorbed into the other operation being performed by the mixnet: the shuffle. This is because both the shuffling and the decryption can be viewed as permutations when the number of messages is small, and can be combined into one essential permutation. Another way of viewing this is to consider the vote for each candidate in a ballot (a mark or no mark) as a separate entity that travels independently through the mixnet (as opposed to being part of a ballot or a contest).

Let $N$ be the number of ballots in an election and let $c$ be the number of candidates on a ballot. Consider three tables: $R$ (stands for receipt values) contains coded

[1] While conducting a performance analysis of our java implementation of the Punch-Scan back-end, we tabulated one million cast ballots in under ten minutes using a Dell Inspiron E1505 laptop, equipped with Intel Core Duo 1.73 Ghz and 1 Gb of Ram.

votes; $T$ (stands for tallies and results) contains clear text votes that are countable by anyone; $D$ (stands for decrypt) connects $R$ with $T$. $R$ is a matrix with $N$ rows and $c$ columns; each row represents a ballot. $T$ is a matrix with $c$ rows and $N$ columns; each row represents a candidate. An element $(i, j)$ is either marked or not marked in $R$ and $T$, a mark in table $T$ corresponds to a clear vote for a candidate, while a mark in table $R$ represents an "encrypted" vote. $D$ is a blob with $N \times c$ elements (the number of rows and columns is irrelevant). Figure 5 gives an example of the three tables for an election with six ballots and two candidates.

The tables are connected by two permutations, $\pi_1$ and $\pi_2$. $\pi_1$ connects $R$ (receipts; coded votes) with $D$ (decrypt): $D_k = R_{\pi_1(k)}$, where $k$ is some canonical representation of $(i, j)$; for example, $k = (c-1)i + j$. $\pi_2$ similarly connects $D$ with $T$ (results or tally): $T_k = D_{\pi_2(k)}$. These permutations are constrained to return a mark for a particular candidate in $R$ to a mark for the same candidate in $T$.

For $c$ candidates, the properties of the permutation may be formalized as follows: let $\pi_1$: $[\{0, 1, \ldots, N-1\} \times \{0, 1, \ldots, c-1\}] \rightarrow [\{0, 1, \ldots, N-1\} \times \{0, 1, \ldots, c-1\}]$ be bijective and let $\pi_2$: $[\{0, 1, \ldots, N-1\} \times \{0, 1, \ldots, c-1\}] \rightarrow [\{0, 1, \ldots, N-1\} \times \{0, 1, \ldots, c-1\}]$ be bijective such that no two elements belonging to the same ballot initially (in the same row in the initial set) are mapped by $\pi_1$ and $\pi_2$ to two elements belonging to the same candidate (the same row in the final set):

$$\forall i,\ j, i \neq j \ \textit{having}\ [i/c] = [j/c] \Rightarrow [\pi_2(\pi_1(i))/b] \neq [\pi_2(\pi_1(j))/b] \tag{4}$$

where $[x]$ represents the greatest integer less than or equal to x. Note that no group operation (such as modulo addition or permutation composition) is performed on the payload. For $c > 2$, the condition requires that the remainder on division by $c$ (the candidate for a mark or nomark) be preserved; because the rows and columns are reversed, one may then wish to view table $T$ canonically as listed column by column, and $R$ row by row.

The audit checks that one of the two properties hold: $D_i = R_{\pi_1(\mathrm{i})}$ or $D_i = T_{\pi_2^{-1}(i)}$ and that the properties of the two permutations $\pi_1$ and $\pi_2$ hold; more precisely it is statistically checked that both $\pi_1$ and $\pi_2$ are injective functions and that Eq. (4) holds. Because the voting system cannot predict which property will be checked, a successful audit implies that both properties hold with high probability.

If a pointer-based mixnet processes an entire ballot, computing the two permutations requires $2 \times (c \times N) \times \log(c \times N)$ operations (e.g., assuming sorting is used to

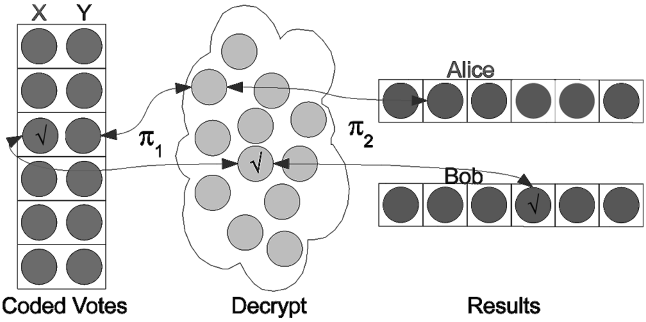

**Figure 5.** Pointer-based mixnet.

**Table 3.** Properties of various mixnets

| | Onion mixnets | Punchscanian mixnet | Pointer mixnet |
|---|---|---|---|
| Distributed authority | Yes | No | No |
| Paths | Dynamic | Static | Static |
| Constructs the ballot | No | Yes | Yes |
| Efficiency | Low | High | Medium |
| Lazy ballot style | Yes | No | Yes |
| Cryptography used | Symmetric and asymmetric encryption | Commitments | Commitments |

generate the permutations), and another $2 \times (c \times N)$ operations for computing the outputs. These operations are for the entire mixnet and for all races.

*Advantages and Disadvantages*

The advantages of pointer mixnets are their efficiency and the possibility of setting up the system before the details of the election are known [10]. Their disadvantage is the central nature of the authority.

Table 3 summarizes the advantages and disadvantages of the three types of mixnet-based decryption mechanisms.

## 6. New Voting Systems

In this section we describe new voting systems formed by combining the front and back-ends of existing systems.

### 6.1. New Voting Systems with Onion Mixnet Back-ends

We first describe the combination of the onion mixnet back-end with the front-ends of PunchScan and Scantegrity.

*PunchScan Ballot with Onion Mixnet*

Recall that $\bigodot_j F_j(seed_j)(\texttt{Ballot}_j) = D(f(s), \texttt{Ballot}_j)$ for the onion mixnet, and $f(s) = \pi_a(s) \circ \pi_{\bar{a}}(s)$ for the Punch-Scan ballot. Hence, for a PunchScan front-end and an onion mixnet back-end, the ballot needs an onion, which will be contained in $x$. The onion contains seeds which will generate permutations whose composition will invert the encrypting permutation $\pi_a(s) \circ \pi_{\bar{a}}(s)$. Thus, in this case, $G$ is the set of permutations on a set of size $c$, and $\odot$ is the composition of permutations. In order to generate the ballot, the ballot manufacturing facility produces a pseudorandom permutation, say $\pi$, as the composition of several pseudorandom permutations $\pi_j$ (as many as there are mixes) each generated from a random seed. $\pi$ is decomposed into two permutations to be used for the two pages of the ballot, by choosing one of the permutations uniformly at random. The seeds for the $\pi_j$ are buried into the onion, which is part of $x$. Decryption involves generating the corresponding $\pi_j$, and performing its inverse. That is, $F_j(seed_j) = \pi_j^{-1}$. This combination requires $N$ onions and $2 \times N$ commitments.

*Scantegrity Ballot with Onion Mixnet*

The best dispute resolution properties for the indication receipts of Scantegrity are obtained when the set of confirmation numbers is large, when a confirmation number is used exactly once among all the ballots, and when the probability of guessing a valid confirmation number of any candidate in a receipt is low. When the Scantegrity ballot is used with the onion mixnet, the Scantegrity front-end generates a set of confirmation numbers for each ballot and a series of operations (as many as there are mixes) that transform the confirmation numbers into candidates names in the end. For example, confirmation number $B7KW8 \rightarrow 3 \rightarrow 7 \rightarrow 1 \rightarrow 1 \rightarrow \mathit{David}$. This can be viewed as a permutation (preceded by a substitution) that is generated, as for the PunchScan ballot, by first constructing as many pseudorandom permutations as there are mixes, and then composing them into a single permutation. The decryption is also similar thereafter; each mix applies the inverse of the pseudorandom permutation corresponding to the seed the mix obtains. This combination requires $N$ onions and $N$ commitments.

### 6.2. New Voting Systems with Punchscanian Mixnet Back-ends

In this section, we describe the use of the punchscanian mixnet with the other front-ends. The essential approach is to absorb the onions into the mixnet and to precommit to both onions and paths. Hence the onions used for the onion mixnet can also be used for the punchscanian mixnet, with two differences: the onions will not be carried with the ballots, and using more than two mixes serves no purpose, as precommitted paths imply that decryption is performed by a single entity. Using a larger number of mixes provides no greater privacy; the use of two mixes is necessary, however, if tally verification consists of RPC.

*Visual Cryptography with Punchscanian Mixnets*

Two punchscanian mixnets are constructed, one for each layer. For each layer of each vote, the pseudorandom component contributed by each mix is committed to, along with the path the ballot layer will travel. The voter's choice of receipt determines the mixnet that will be used to decrypt her vote. From the chosen layer, the pixels that are generated pseudo-randomly are discarded and the other pixels are run through the corresponding punchscanian mixnet. This combination requires $2 \times (2 \times N) + 2 \times N$ commitments (the first term for the back-end, and the last term for the front-end).

*Prêt à Voter with Punchscanian Mixnets*

The onions of the Prêt à Voter ballot are committed to, along with the path the ballot will take, at the mixnet. The onion is not part of the payload. Such a system has been described in detail by van de Graaf in [26]. This combination requires $2 \times N + N$ commitments.

*Scantegrity with Punchscanian Mixnets*

The same procedure is followed as for Scantegrity with the onion mixnet, except that the onion is not carried with the ballot, and is committed to in the appropriate mixes along with the pre-computed path the ballot will take. This combination requires $2 \times N + N$ commitments.

### 6.3. New Voting Systems with Pointer Mixnet Back-ends

*Visual Crypto with Pointer Mixnet*
If each pixel in the vote image is treated as independent of all other pixels, each pixel is equivalent to a candidate (from the point of view of the back-end). A direct application to the pointer-based mixnet would result in a very inefficient system, as the number of commitments needed for the back-end would be twice the number of pixels in the image.

*Prêt à Voter with Pointer Mixnet*
Each candidate, and thus mark position, is treated independently and its path and ending point in the table with the clear votes are committed to, just as with Scantegrity. This combination requires $2 \times (c \times N) + N$ commitments (the first term for the back-end and the last term for the front-end).

*PunchScan with Pointer Mixnet*
Each position that can be marked is treated independently and its path and ending point in the clear vote table is committed to, just as with Scantegrity. This combination requires $2 \times (c \times N) + 2 \times N$ commitments (the first term for the back-end and the last term for the front-end).

## 7. Conclusions

We have presented a unified view of four practical, end-to-end, voter-verifiable voting systems that have been proposed recently as monolithic blocks. We present a concrete separation between the way the ballot is presented and how the voters interact with the system (the front-end) and the way the ballots are decrypted and the tally is verified (the back-end). We present the properties of these front and back-ends, and describe simple ways to combine them. This gives great flexibility in the choice of a voting system for a particular jurisdiction that values some properties more then others (e.g., privacy more than usability). Our work opens a new way of looking at future voting systems, component-wise. An immediate benefit is the possibility of designing accessible front-ends for each of the presented back-ends, as opposed to trying to mimic the front-end in an accessible manner. Further research can focus only on improving or changing a particular component of a voting system (e.g., back-end), as long as it can interact with the other component (e.g., front-end).

## Acknowledgments

The authors would like to thank David Chaum, Jonathan Stanton, Aleks Esses, Rick Carback and Jeremy Clark for inspiring discussion, Geanina Popoveniuc, Gedare Bloom and Eugen Leontie for reviewing early versions of this work, and an anonymous reviewer for several useful suggestions. Both authors were also supported in part by NSF grant 0831149.

## About the Authors

Stefan Popoveniuc has successfully defended his doctoral dissertation at The George Washington University. His areas of interest are computer security and privacy in general, and electronic voting in particular. He has designed and built four complete voting systems, and he is a member of the Punchscan team, which won first place at the 2007, intercollegiate voting systems competition, VoComp. Popoveniuc has a BS in computer science from the Politechnical University, Bucharest, Romania.

Poorvi L. Vora is Assistant Professor in the Department of Computer Science at The George Washington University. In 1986, she received the BTech degree in electrical and electronics engineering from the Indian Institute of Technology, Bombay, India. In 1988 and 1993, respectively, she received the MS and PhD degrees in electrical engineering from North Carolina State University, Raleigh, NC, USA; and in 1990, she received the M. S. degree in mathematics from Cornell University. Her areas of interest are electronic voting and cryptology.

## References

1. Adida, B. 2006. Advances in Cryptographic Voting Systems, PhD thesis, MIT.
2. Adida, B. June 2008. Helios: Web-based open-audit voting. In *Proceedings of the Seventeenth Usenix Security Symposium (USENIX Security 2008)*, Sanjose, CA. Berkeley, CA: USENIX Association.
3. Adida, B. and R. L. Rivest. 2006. Scratch & vote: Self-contained paper-based cryptographic voting. In WPES '06: Proceedings of the 5th ACM Workshop on Privacy in Electronic Society, New York, NY: ACM Press, pp. 29–40.
4. Benaloh, J. 2006. Simple verifiable elections. In EVT '06: Proceedings of the USENIX/ Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop, Berkeley, CA: USENIX Association, p. 5.
5. Chaum, D. 2007. Private communications.
6. Chaum, D. 2003. U.S. patent 7210617. Filed 21 January 2003; Issued 1 May 2007 – Secret-ballot systems with voter-verifiable integrity.
7. Chaum, D. January/February 2004. "Secret-Ballot Receipts: True Voter-Verifiable Elections," *IEEE Security and Privacy*, 2: 38–47.
8. Chaum, D. Recent results in electronic voting. In Presentation at Frontiers in Electronic Elections (FEE 2005), Milan, Italy, September 2005. ECRYPT and ES-ORICS. Available online at http://www.win.tue.nl/nberry/fee2005/
9. Chaum, D., R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman. 2008. Scantegrity ii: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In EVT '08: Proceedings of the USENIX/Accurate Electronic Voting Technology on USENIX/ Accurate Electronic Voting Technology Workshop. San Jose, CA. Berkeley, CA: USENIX Association.
10. Chaum, D., A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. T. Sherman, and P. Vora. May/June 2008. "Scantegrity: End-to-end Voter Verifiable Optical-Scan Voting," *IEEE Security and Privacy* 6: 40–46.
11. Chaum, D., P. Y. A. Ryan, and S. Schneider. 2005. A practical voter-verifiable election scheme. In *ESORICS, Volume 3679 of Lecture Notes in Computer Science*, edited by S. De Capitani di Vimercati, P. F. Syverson, and D. Gollmann. Heidelberg: Springer Berlin, pp. 118–139.
12. Chaum, D., J. van de Graaf, P. Y. A. Ryan, and P. L. Vora. 2007. Secret ballot elections with unconditional integrity. Technical report, Cryptology ePrint Archive, Report 2007/270.
13. Chaum, D., B. Hosp, and S. Popoveniuc. 2009. "Accessible Voter Verifiability," *Cryptologia*. 33(3): 283–291.

14. Chaum, D. L. 1981. Untraceable mail, return addresses, and digital pseudonyms. *Commun. ACM* 24(2): 84–90.
15. Choonsik Park, K. I. and K. Kurosawa. 1994. Efficient anonymous channel and all/nothing election scheme. In *Advances in Cryptology—EUROCRYPT '94*. Heidelberg: Springer Berlin.
16. Furukawa, J. and K. Sako. 2001. An efficient scheme for proving a shuffle. In CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, London, UK: Springer-Verlag, pp. 368–387.
17. Jakobsson, M., A. Juels, and R. L. Rivest. 2002. Making mixnets robust for electronic voting by randomized partial checking. In *Proceedings of the 11th USENIX Security Symposium*, Berkeley, CA: USENIX Association, pp. 339–353.
18. Lundin, D. June 2007. Component based electronic voting systems. In IAVoSS Workshop On Trustworthy Elections (WOTE 2007), Ottawa, Canada: University of Ottawa.
19. Naor, M. and A. Shamir. 1995. "Visual Cryptography," *Lecture Notes in Computer Science LNCS*, 950:1–12.
20. Neff, C. A. 2001. A verifiable secret shuffle and its application to e-voting. In 8th ACM Conference on Computer and Communications Security, pp. 116–125. Philadelphia, PA: ACM Press.
21. Popoveniuc, S. and B. Hosp. June 2006. An introduction to PunchScan. In IAVoSS Workshop On Trustworthy Elections (WOTE 2006), Robinson College: Cambridge UK.
22. Popoveniuc, S. and P. L. Vora. July 2008. A framework for secure electronic voting. In IAVoSS Workshop On Trustworthy Elections (WOTE 2008), Leuven, Belgium.
23. Popoveniuc, S. 2009. A framework for secure mixnet based electronic voting. Ph.D. Thesis. Available online at http://gradworks.umi.com/33/44/3344934.html
24. Ryan, P. Y. A. and S. A. Schneider. 2006. Prêt à Voter with Re-encryption Mixes. In *ESORICS, Volume 4189 of Lecture Notes in Computer Science*, edited by D. Gollmann, J. Meier, and A. Sabelfeld. Heiledberg: Springer-Verlag, pp. 313–326.
25. Sako, K. and J. Kilian. 1995. Receipt-free mix-type voting scheme: A practical solution to the implementation of a voting booth. In *Advances in Cryptology – EUROCRYPT '95*. Saint-Malo, France.
26. van de Graaf, J. 2007. Merging Prêt à Voter and PunchScan. Cryptology ePrint Archive, Report 2007/269. http://eprint.iacr.org/2007/269.pdf
27. Vora, P. L. 2005. David Chaum's voter verification using encrypted paper receipts. Cryptology ePrint Archive, Report 2005/050. http://eprint.iacr.org/