A FRAMEWORK FOR SECURE MIXNET-BASED ELECTRONIC VOTING

by
Ştefan POPOVENIUC

B.S. June 2004, Universitatea Politehnica Bucureşti, România
A Dissertation submitted to

the Faculty of
the School of Engineering and Applied Sciences
The George Washington University
in partial fulfillment of requirements
for the degree of Doctor of Science

May 17, 2009

Dissertation Directed by
Poorvi Vora
Assistant Professor of Engineering and Applied Science
and
David Chaum
Cryptographer

The School of Engineering and Applied Science of The George Washington University certifies that Stefan POPOVENIUC has passed the Final Examination for the degree of Doctor of Philosophy as of December 10,2008. This is the final and approved form of the dissertation.

A FRAMEWORK FOR SECURE MIXNET-BASED ELECTRONIC VOTING

Ştefan POPOVENIUC

Dissertation Research Committee:

Poorvi Vora, Assistant Professor of Computer Science, Dissertation Co-Director

David Chaum, Cryptographer, Dissertation Co-Director

Rachelle Heller, Professor of Engineering and Applied Science, Committee Member

Julie J.C.H. Ryan, Assistant Professor of Engineering Management and Systems Engineering, Committee Member

Rhys Price Jones, Professor of Computer Science, Committee Member

Rahul Simha, Professor of Engineering and Applied Science, Committee Member

Abstract

A FRAMEWORK FOR SECURE MIXNET-BASED ELECTRONIC VOTING

Public sector elections are too important to be carried out without voter oversight. Voters should have the ability to audit every step of the process, from the recording of the cast ballots, to the tallying of the votes and the declaration of the election outcome. This is a challenging problem, because the voting system must not infringe on voter privacy; coerced voters do not contribute to the advancement of democracy and the possibility of coercion should be explicitly addressed by any modern voting system. Current commercially-available voting systems do not provide voters the ability to verify that votes are indeed tallied as cast.

This dissertation explores a new class of voting systems that have unexpected properties in all aspects of the voting process. It presents novel techniques used in the implementation of verifiable, secret-ballot voting systems that are secure, fast, reliable and easy to use and understand. These techniques are presented in the context of an original, complete, framework for secure electronic voting that allows the integration of many new techniques into a coherent and complete voting system. The framework encompasses both polling place voting and remote voting (including Internet voting) and provides a large variety of choices in terms of security, usability, availability, computational effort and cost. A variety of modular components are proposed, that can be combined in virtually any configuration to serve the concrete purpose of particular elections and organizations. The dissertation also describes three elections that were carried out with the implemented voting systems.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The voting process is a key feature of a democracy, and the extent to which it is accurate, fair and incorruptible determines the health of the democracy it supports. For this reason, voter intentions should be collected and aggregated in a completely transparent manner to obtain the election outcome. On the other hand, because knowledge of an individual's vote can lead to vote buying and coercion, the process that records voter intentions and tallies them should also maintain ballot secrecy. This leads to a seemingly contradictory set of requirements: a completely transparent system that maintains ballot secrecy. A quarter century of cryptography has led to a novel class of systems known as *end-to-end independently verifiable (E2E)* systems that enable a voter to determine that her vote was recorded correctly, and any observer to determine that recorded votes were correctly tallied. This dissertation describes original contributions to the literature on *E2E* systems; contributions of this dissertation are in the development of new voting system designs, in their theoretical analysis, in their implementation, and in their use for real binding elections. In particular, this dissertation describes (a) the design, implementation and use of *PunchScan*, the first *E2E* system to be used in a binding election, (b) the design and implementation of *Scantegrity*, an *E2E* system that has been recommended for use in the Takoma Park municipal elections of 2009 by the Takoma Park Board of Elections to the Takoma Park Council [CM08] (c) a framework for all *E2E* systems that use a particular type of component, known as a mixnet, and (d) proofs of the properties of *PunchScan*. It also present a novel approach to the design of remote voting systems, where voters cast their votes from a remote location, not from a polling booth.

1.1 Background

Ideally, every voter should be able to check that votes were tallied correctly. If the votes are public, this is trivial to do. Everyone knows how everyone else voted; if a vote is recorded incorrectly, this can easily be challenged and corrected. Such an approach, would, however, enable coercion and vote buying, and, as a result, would not result in an aggregate of the true opinions of voters. In the modern world, the risk of coercion and vote buying is primarily addressed through the use of a secret ballot. Thus, individual preferences remain secret, and only their aggregation becomes public. One of the most documented examples of the use of secret ballots is the election of the Pope, which has been conducted using a secret ballot since 1288. The Australian ballot [Wike], introduced at the end of the nineteenth century, is a special type of secret ballot, closest to that used in most elections today. It has all the names of the candidates pre-printed on it; a voter votes by making a certain type of mark (e.g. the voter fills in an oval) next to the candidate's name.

From a historical point of view, voting has moved from centralized voting in a single *Town Hall*, to distributed polling places, and more recently, to remote voting. When all the voters are in a single location and the voting process lasts a short period of time (minutes or hours), secret voting is not a big problem. All voters and observers can watch the process of vote tallying and verify that all steps are conducted honestly. Disputes can be resolved on the spot. David Chaum has coined the term *room voting* to describe this process. As the voting process became more popular, and as the voters spread out over larger geographical areas, it became impractical to call everyone to the central *agora*. Separate polling places were organized across entire countries, the counting process was local to each polling location, and the result tabulation was centralized. Obvious problems appeared with this approach; an individual in Athens cannot check the partial results in Constantinople. In modern days, with tens of thousands of polling places in a country of average size, and with hundreds of voters casting votes over many hours (typically twelve to sixteen), it is impossible to require voters to assist in the vote counting process, to have one representative of every interested organization in each polling place, and to compute and centralize fast and accurate results while preserving the confidence and user-friendliness of the voting process. Even worse, with the introduction of Direct Recording Electronic (DRE) machines (without Voter Verified Paper Audit Trails (VVPAT)), voters or election officials cannot count the ballots themselves; instead, a private vendor does it for them using a “voting machine”.

The use of cryptography provides a resolution of the apparent paradox of a transparent voting system that enables voting in several, geographically distant polling locations, while providing ballot secrecy. In the modern cryptographic model, voting is a multiparty computation [Yao82], where several parties each provide private input to a computation, and all must be convinced that the result of the computation is correct. It is known that a multiparty computation cannot guarantee both the correctness of the result and the secrecy of the inputs without assumptions about the adversary or of the manner in which the computation is performed. There are, however, constructive proofs that demonstrate the possibility of achieving the goals of multiparty computation when one of the adversaries—either the adversary trying to obtain information on the inputs, or the adversary trying to falsify the output of the computation—is computationally bounded, and certain types of secure cryptographic primitives exist. Thus, in the context of the voting problem, a voting system has one of the following vulnerabilities. In one type of system, an adversary who has access to computational power and time sufficient to break cryptographic primitives can change the outcome of the election without detection, but cannot determine any information about individual votes. In the other type of system, an adversary who has access to computational power and time sufficient to break cryptographic primitives can determine how individual voters voted, but cannot affect the election outcome once votes are cast. In practice, one may assume that reasonably secure cryptographic primitives exist and neither type of attack is possible; after all, practical cryptographic primitives are secure enough to be used on a daily basis for financial transactions. The last two decades have seen the creation of theoretical voting systems that work in principle, and have desirable properties that can be mathematically proven. However, hardly any of the systems were ever used in practice, and very few ever had constructed prototypes. Their practicality and performance were never addressed or clarified.

A more recent trend in voting is the use of remote voting in an attempt to increase voter turnout. Voter turnout is one of the most important measures of the authority of an elected leader. For this reason, and for many others, increasing voter turnout is a crucial goal for candidates, a democratic society in general, and for the officials organizing the elections. A solution that was believed to increase voter turnout was the introduction of remote voting. There remains controversy among political specialists over the beneficial impact of introducing remote voting, as some studies concluded that voter turnout does not increase to a statistically significant extent with the use of remote voting [KM06]. Also, the studies showed remote voting did not decrease the cost of running elections [Dun], nor the tallying time [Lan]. However, considering the current trend of decentralization, it seems that the future of elections includes remote voting, with the probable technical support of computers and the Internet. A key problem in remote voting is improper influence: a coercer can sit next to a voter and force him to mark the ballot in a certain way. If computers are involved in the casting process, a second fundamental problem is that the computer may be infected with viruses that can manipulate the votes in favor of certain candidates, while preserving the appearance to the voter that everything is as expected.

1.2 Related Work

Recent advances in cryptographic voting systems have led to the invention of several *E2E* voting systems [Cha04, CRS05, PH06, CCC$^+$08]. These systems provide (encrypted) receipts to the voters that can be used to ensure that their vote is counted, without revealing the vote. Voters can later check that their receipts form part of the electoral record (the virtual ballot box) and anyone can verify that the voting system correctly decrypts the entire electoral record. The verification does not reveal the vote of individual voters. These systems empower voters and allow them to play a key role in the auditing of the results that the election officials make public.

Before 2003, there was little work on end-to-end voter verifiable systems; one notable exception is the work of Andy Neff [Nef01a]. In 2003, David Chaum proposed a voting system that was based on visual cryptography [Cha04]. This system was probably the first to allow voters to verify the encryption of their votes without having access to trusted computation in the voting booth, and at the same time to be able to prove with high probability, that the voting system was not cheating in encrypting the vote. The ballot form, however, was very complicated and presented a very unfamiliar user interface. Shortly after 2003, Peter Ryan proposed a simplification of the ballot, leading to Prêt à Voter [CRS05]. This was followed by a sketch of *PunchScan* in talks by David Chaum [Cha05a]; *PunchScan* proposed a different ballot format, as well as a simpler and far more efficient way of obtaining the tally. Except for Chaum's first system, dubbed *Citizen-Verified Voting (CVV)*, [Cha04], implemented by Ben Hosp and others at GW, none of the voter-verifiable systems were implemented before the work of this dissertation. At about the same time as the work in this dissertation, several new *E2E* systems were invented, notably Scratch and Vote [AR06]; also, the international voting system competition motivated the implementation of two other *E2E* voting systems [oT07], [oS07].

1.3 Contribution of this dissertation

The key contributions of this dissertation are in the general area of *E2E* systems that use components similar to mixnets [Cha81]. First, the dissertation takes a fresh look at the recent voter verifiable voting systems based on mixnets and presents a single general voting system, consisting of a *back-end* and a *front-end*. It demonstrates that the four most interesting such systems (CVV, *Prêt à Voter*, *PunchScan* and *Scantegrity*) are specific instances of the general system. Combining the distinct back and front-ends of the existing systems with one another leads to twelve distinct voting systems. Additionally, the systems can now be examined more generally, and experiences from the implementation of one system can be generalized to others.

Second, this dissertation looks in great detail at the integrity and privacy properties of the existing systems, making formal statements and providing proofs. Third, it describes voting system designs, presenting new front and back-ends that bring hitherto unachieved properties, and shows how they compare to the previously existing ones. For the *PunchScan* back-end, new techniques are proposed that address aspects like the generalization of the back-end, contest partitioning, recovering from failed audits. The dissertation also examines the influence of the configuration parameters on the integrity and privacy provided, presents a new vulnerability in the way the integrity checks are performed and a way to mitigate it. Fourth, the dissertation describes the implementation of all the front and back-ends by the author, and includes a performance analysis. This work is particularly interesting, because it includes a description of the implementation of *PunchScan*, the first *E2E* system to be used in a binding election. Fifth, it describes two binding elections using the *PunchScan* voting system, the experience of the organizers and voting officials and how easy easy was it for them to learn how to use the system. Finally, the thesis proposes a new back-end that can be used for remote voting, allowing the voter to securely cast a ballot from a computer that is potentially infected with malware.

1.3.1 Collaboration and authorship

Like any other doctoral student, all the work has been done in collaboration with the supervisor of the thesis: Jonathan Stanton, who left the computer science department at GWU in the summer of 2008, Poorvi Vora, who headed the security research group at GWU became my advisor after Prof. Stanton relocated, and David Chaum, who closely supervised the development of the PunchScan and Scantegrity voting systems. Besides the academic and research advisors, I have had the opportunity to collaborate with faculty from other universities: Ronald Rivest from MIT, Allan Shernam from University of Maryland Baltimore County and Peter Ryan from Newcastle University. I also had the opportunity to collaborate with graduate students from various American, British and Canadian universities: Rick Carback from University of Maryland Baltimore County, Aleks Essex from University of Ottawa, Jeremy Clark from Waterloo University, Emily Shen from MIT, Ben Hosp from GWU and David Lundin from University of Surrey. I have coauthored academic papers with all of the above collaborators.

An early version of the detailed description of the *PunchScan* back-end (see Sect. 5.2) was initially presented in a paper co-authored with Ben Hosp and presented at WOTE 2006 [PH06]. The formal proofs of integrity and privacy and much of the formalization presented in Sect. 5.2 are presented for the first time in this dissertation. The first election using a verifiable voting system (see Sect. 7.1) was carried out in collaboration with (in alphabetical order) Rick Carback, David Chaum, Jeremy Clark, and Aleks Essex. The contest partitioning technique presented in Sect. 5.2.9 was inspired by this election, and the research was performed in collaboration with Jonathan Stanton.

The *Scantegrity* front-end (see Sect. 6.2) was developed in collaboration with Rick Carback, David Chaum, Jeremy Clark and Aleks Essex, and presented by me at the Dagstuhl meeting on Frontiers of Electronic Voting [PCE$^+$08]. Later, this was published in IEEE Security and Privacy [CEC$^+$08]. The pointer based back-end, also unique to *Scantegrity*, was first presented in the same paper. A modification of the *Scantegrity* front-end led to the *Scantegrity II* front-end (see Sect. 6.2.1.1) [CCC$^+$08], which was developed in collaboration with Rick Carback, David Chaum, Jeremy Clark, Aleks Essex, Ronald Rivest, Peter Ryan, Emily Shen, Alan Sherman.

The technique used to make mail-in ballots as secure as ballots cast at polling places, presented in Sect. 6.5.2, was developed with David Lundin, and presented at VOTE-ID 2007 [PL08]. The technique for constructing a virus-proof front-end was developed in collaboration with Poorvi Vora, and presented at WISSec 2008 [PV08b]. The safe auditing technique using RPC (see Sect. 5.2.7) was developed independently and is currently submitted for publication.

The framework for mixnet-based voting systems was developed in collaboration with Poorvi Vora.

1.4 Organization

This dissertation is organized as follows: Chapter 2 contains the basic definitions, and a description of the problems addressed by this dissertation. Chapter 3 starts with a brief history of voting technology, continues by presenting the three main modern techniques in the research area of provably secure voting systems. It also contains descriptions of the systems that existed before this dissertation, and upon which this dissertation is built. Some general discussion on DREs, as well as background work in remote voting is also presented.

The contributions of this dissertation begin with Chapter 4, which presents a framework for mixnet-based *E2E* systems, which can be viewed as a single general mixnet-based *E2E* voting system. The chapters 5 and 6 present in detail the back-ends and the front-ends of voting systems that were previously viewed as monolithic. The *PunchScan* back-end is extensively analyzed, with proofs of integrity and privacy, a generalization of the back-end, and techniques that address real world scenarios. The pointer based back-end is presented as a particular case of the *PunchScan* back-end, thus reusing all the proofs and techniques.

The *PunchScan* and *Scantegrity* front-ends are presented as particular cases of symmetric and asymmetric ballots, respectively. General proofs of the integrity and privacy of the symmetric and asymmetric ballots are described. In the same chapter, Chapter 6, for each front-end, a simple way of connecting it to the existing back-ends is described. Two special cases of front-ends are presented at the end of Chapter 6, one front-end that is accessible for visually impaired voters and one that offers the same level of integrity as the polling place setting for mail-in ballots.

Two case studies of the elections we have run using the implementation of our secure design are presented in Chapter 7, along with some problems that we have faced and the lessons we have learned; Chapter 8 presents a front-end that can be use on remote computers that are potentially infected with malware. Conclusions and future work are presented in Chapter 9.

Chapter 2

The Voting Problem - preliminaries

The voting problem may be viewed as a particular instance of the more general problem of computing an arbitrary function of several variables. Such an approach allows for the reuse of important concepts and results proven for the broader setting. On the other hand, some impossibility results for the general case may not be valid for the particular problem of voting. This chapter introduces both general and voting-specific definitions that will help in the understanding of the specific challenges of electronic voting. It introduces and defines the notions of integrity, privacy, software independence and end-to-end security. While most of the definitions are from the literature, the definitions of the privacy of a ballot, the average privacy, voluntary and involuntary privacy as well as the definition of integrity are original contributions of this dissertation.

2.1 Defining what a voting system is

This section provides a list of properties a voting system may reasonably be required to possess, and derives a general definition of a voting system. The central correctness properties of voting systems are presented first, followed by some practical and social requirements and finally by procedural and human factor requirements.

For a computer scientist, voting is a particular instance of a secure multiparty computation, where the computation is, for example, the simple arithmetical addition of votes for individual candidates.

Definition 1 (Multi party computation(MPC)[GMW87])**.** *Given a set of participants*

$$\mathbf{P} = \{\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, ..., \mathbf{p_n}\}$$

each having private data

$$\mathbf{D} = \{\mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3}, ..., \mathbf{d_n}\}$$

respectively, compute a public function F from the set $\mathbf{D}$.

Definition 2 (Secure MPC)**.** *A multiparty computation is called secure if and only if no participant or outsider can learn more then the result of the function from all the eventual interactions that take place and the result of the function can be independently, and eventually universally, verified.*

The voting problem is a specific multiparty computation problem, where the function F determines the outcome of the election, and the private data consists of the votes. For example, F may be a simple count of identical elements among $\{\mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3}, ..., \mathbf{d_n}\}$, or a more complex function for more complex tallying methods.

Definition 2 specifically refers to the two most central properties of voting systems: integrity and privacy. On the one hand, nothing but the result of the function should be available for any one entity. On the other hand, the result the function returns should be the true result of the function, as if all the operations were performed publicly, and none of the input data were private.

It is easy to see that the voting problem is trivial if only one property, privacy or integrity, was required. An example of a voting system that has perfect integrity is public voting: each voter posts her vote publicly and everyone can check all the votes. This voting system has no privacy whatsoever. At the other extreme, a voting system that collects all the votes secretly, then burns the unopened ballot box and declares arbitrary results, has perfect privacy; the votes cannot be hidden any better than this. However, providing both properties is considerably more challenging. It has been shown that information-theoretic constraints make it impossible for a voting system that does not depend on physical properties (such as: this ballot box does not have a physical trapdoor) to possess both integrity and privacy if the adversary is computationally unbounded [HV08]. Relaxing the requirements does, however, lead to constructive results. Voting systems with both integrity and privacy have been demonstrated in the literature when one of the two adversaries (the integrity adversary or the privacy adversary) is computationally bounded [CvdGRV07, MN07].

On applying definition 2 to the voting problem, we see that a voting system provides privacy if the only new information available on individual votes after the election is contained in the election results. That is, the eventual proofs, or any other records that come out of the voting system, do not leak any additional information, beyond that contained in the tally, about who voted for whom. The importance of quantifying privacy was discussed in an article in *ACM Risks*[VAB$^+$04] and the definition of privacy provided below was formalized in [CHVW05].

Definition 3 (Privacy [CHVW05])**.** *An election system is perfectly private if an adversary's information about a voter's ballot choice(s), as obtained through the election technology and process/procedures, is not affected by the actual vote cast by the voter on election day.*

Information regarding education, income, and geographical location, combined with the tally, can be used to improve estimates of the choices of a specific voter. However, this privacy leakage occurs because the tally is revealed, and because voters provide information about their inclinations through pathways other than through the voting systems used. Such leakage occurs no matter what specific voting system is used, therefore it does not contribute to any privacy loss with regards to definition 3. In contrast, records indicating the order (total or partial) in which the votes were cast can leak information and thus can lead to a reduction in privacy. We now describe formal measures of the extent of privacy loss in voting systems that do not provide perfect privacy.

Definition 4 (Privacy set)**.** *Let $F : \mathbf{A} \rightarrow \mathbf{B}$ be a function, where A and B are two finite sets. Let $\mathbf{b} \in \mathbf{B}$. The privacy set of $\mathbf{b}$ is the set $\mathbf{S} = \{\mathbf{x} \in \mathbf{A} : P(F(x) = \mathbf{b}) > 0\}$, where $P(x)$ is the probability of event x.*

The privacy set of an element is the set of elements from which it could have come. Some of the mappings may be known, along with properties of the function F. Let F be a random bijective function and $F(\mathbf{a}) = \mathbf{b}$. Then the privacy set of $\mathbf{b}$ has cardinality 1, because it is only the element $\mathbf{a}$; the cardinality of any element $x \neq \mathbf{b}$ is $|\mathbf{B}| - 1$, where $|\mathbf{B}|$ is the cardinal of $\mathbf{B}$, since a single mapping is known and all the other mappings are possible (because f is random).

Definition 5 (Privacy of a ballot)**.** *The privacy of a cleartext cast ballot is the cardinality of its privacy set.*

Definition 6 (Average privacy)**.** *The average privacy of a voting system is the arithmetic average of the privacy of each cast ballot.*

Definition 7 (Involuntary privacy)**.** *The involuntary privacy of a ballot is the privacy of a ballot measured when the voter provides all the data she has access to about her ballot.*

If a voter is constructing the ballot herself, as in [Cha82, Cha81], then the voter has access to the randomness used for creating it (blinding factors, random padding) and can provide all the data to a coercer who can check that the ballot is correctly formed. This way a voter can convince a coercer that she voted a certain way. A voter may be tempted to voluntarily give out the data (for a reward), or may be coerced to do it, under the threat of punishment.

Definition 8 (Voluntary privacy)**.** *The voluntary privacy of a ballot is the privacy of a ballot when the voter does not provide any information to help reduce the privacy of her ballot.*

Under definition 8, a coercer only has access to public data, whereas under definition 7 a coercer has access to both public data and data supplied by the voter. We now turn to the definition of integrity.

Definition 9 (Integrity)**.** *Let* $\mathbf{V} = \{\mathbf{v_1}, \mathbf{v_2} ... \mathbf{v_n}\}$ *be the set of the intentions of the* n *voters that are allowed to vote, let* F *be a tally function, let* T_{ideal} *be the tally obtained when applying* F *to* V *(the true tally of the intentions), and let* $T_{announced}$ *be the the official results of the election. Integrity means that* $T_{ideal} = T_{announced}$*. In other words, all the votes were tallied as intended.*

Integrity of a voting system refers to computing the final tally from the true intention of the voters. There should not be any modification or interpretation of the voter intention, nor any mistabulation. Votes should not be changed, injected, deleted, forgotten or not counted. In other words, integrity can both make sense to the persons who are directly involved in all the steps of the process, and can a property that anyone can verify. For an in depth dissection on how to achieve integrity in a voting system, see Sect. 2.4 and Sect. 2.3. Integrity does not capture eligible voters that do not cast a ballot for various reasons, such as registration problems, failure to receive a ballot, or a voter failing to submit or mail it in. But it does capture voters that intentionally cast a blank vote.

A voting process is *transparent* if anyone can check the correctness of the tally. If only entities that have certain privileges (e.g. the election officials) can check the tally, or only partial (e.g. precinct) tallies can be checked, then the process is not transparent.

Loosely speaking, a property provided by a protocol is said to hold *unconditionally* (or information theoretic), if anybody with access to unbounded computational power cannot cheat the protocol and prove a different property. A property provided by a protocol is said to hold *in the computational model*, if anybody with access to polynomial time and memory cannot cheat the protocol and prove a different property.

Having defined integrity and privacy we return to the fact that it is not possible to achieve both when the adversary is computationally unbounded. When a choice has to be made between a voting system that has unconditional integrity and computational privacy (example of such voting systems are [Cha04], [CRS05]) and one that has computational integrity and unconditional privacy (e.g. [MN07], [CFSY96]), one has to determine which of the two properties is more important.

The case for unconditional integrity is perhaps clear. When the integrity is unconditional, one is ensured that the official tally is the real one, and there is no way anyone could have changed it, even with access to unlimited computational resources, or, more likely, with access to a solution to a problem believed to be difficult in the computational model under consideration. Also, in practice, voters are typically more concerned about integrity then about privacy. This is illustrated by the fact that remote voting is very popular (according to a PEW report [PEW08] more than 38 million ballots - about 25% were cast by mail); allowing a voter to vote from anywhere allows him or her to prove his or her vote to anyone present while the ballot is being filled.

The case for unconditional privacy is more subtle. If hard problems protecting the integrity of an election are solved after the election results are announced, the adversary obtains no advantage. On the other hand, if hard problems protecting the privacy of the votes are solved after the election is announced, the adversary may still launch a successful coercion attack. And a coerced election has philosophically no integrity, since, even though the totals were computed correctly, the votes do not capture the true intent of the voters.

In some voting systems (e.g. [Cha04, MN07]) the choice between unconditional privacy and unconditional integrity is made by the design itself. Other voting systems, such as some described in this dissertation, allow the user to make the choice. If best current practices are in use [LV01], it is unlikely that a computationally bounded adversary will be able to break the encryption used in voting systems.

All the systems described in this dissertation make use of a bulletin board to make available information for the purposes of verifying the tally. When the system makes available information on a secure public bulletin board, we say it *publishes* the information. We now formally define it.

Definition 10 (Secure public bulletin board (SPBB))**.** *A secure public bulletin board* *is a place holder for information such that*

1. *(Public) anyone is allowed to read the information*
2. *(Append only) no record can be deleted or modified*
3. *(Consistent) all the records are read as they were posted*

While this dissertation focuses on integrity and privacy of voting systems, they possess several other required properties, the most relevant of which are enumerated here:

- Usability - The procedures necessary for a successful election with the voting system should be easily explainable and understandable by both the personnel in charge of running the election (the election officials) and by the voters themselves. This does not necessarily require that the inner mechanisms of the voting system be easily understandable by everyone (much as knowledge of the Carnot cycle is not necessary to operate a car).
- Accessibility - no major modifications or architectural changes should be necessary to adapt the system for voters with special needs, such as voters with visual impairments, voters with poor member control, etc. If the voting system produces cleartext ballots at its output, a desirable property implied by the privacy property is that the cleartext ballots cast by voters with special needs be indistinguishable from those cast by other voters.
- Flexibility - the system should support all the known ways to capture and aggregate the intent of the voters (see Sect. 3.3). From the simple binary vote (choose one out of two) and first past the post (choose one out of n), to plurality and approval voting (choose m out of n), rank voting, range voting, write-ins, etc.
- Robustness - the system should be able to function properly in common potential failure cases such as loss of power, floods or fire hazards.
- Scalability - the system should be able to handle all the votes in a country, if the entire active population votes.
- Cost effective - the cost per vote should be kept low. This cost includes purchasing the system initially, maintaining and upgrading it, running it, training the people to use it, disposing of it.

- Speed - The voting system should produce the tally in a timely manner. Additionally, the ballot casting process should also be efficient, with a small number of steps in the voting ceremony. It should not require the voter to wait too long for the voting equipment.

- Ecological - while operating, the voting system should have a minimal impact on the environment. The replacement of one generation of the voting system with the next generation should be environmentally friendly.

- Legal - the state or federal laws should not have to be changed to explicitly allow (or not explicitly forbid) specific aspects of the voting system, or the entire voting system.

2.1.1 What a voting system is not

In the mechanics of an election, besides the actual vote casting methodology, other problems can appear. These are briefly mentioned here. The focus of the dissertation is not on these problems, however, any designer of new and innovative techniques should bear these in mind.

An important problem not addressed in this work is that of voter registration. The question of who is allowed to vote is a political one, but the process of registration may pose significant obstacles for voters and therefore decrease voter turnout. Examples of less obvious aspects of the "who is allowed to vote?" question are related to:

- Whether people with multiple residencies are allowed to vote for only one of the jurisdictions, or for each one where they own property.

- Whether students, who are temporarily dislocated, are allowed to cast a vote for their temporary district or permanent district.

- What the exact time frame and procedure for registration is.

- Whether it is possible to register once for many elections.

- What human and technical means are required, and used, to keep the list of active voters up-to-date, including voters that moved, lost or gained or regained their right to vote.

In the case of mail-in or absentee ballots even more details need to be fleshed out, especially with respect to the exact detailed process for mailing and receiving the ballots, for transcribing the received ballots, and for unfilled ballots returned as undeliverable. To make the problem of absentee ballots even more complicated, it should be mentioned that approximately six million potential American voters live overseas. The OUCAVA report of 2007 [Com07] indicates that only one million voters request a ballot, and only around three hundred thousand cast a valid one. This represents a 5% turnout of absentee overseas voters. The main problems arise because proper return postage is not included, because military post office services are used and the delivery time to conflict zones is typically large, and because it is a challenge to deliver ballots on submarines or in places so remote that the only means of communication is electronic.

In the context of precinct-based voting there are managerial issues when it comes to installing the certified voting application on the voting machines, configuring the machines for the current elections, testing them, securing them physically, distributing the machines to the polls, training the personnel to use them, creating posters on how the voters should use them and what should be done in case of a failure. Also, from an election administration point of view there is always the discussion on how to interact with the manufacturers in terms of deliverables, regular maintenance, and updates of software and/or hardware.

These are all valid, interesting practical question, but this work does not address them.

2.2 Current electronic voting systems

We present a general model of any Direct Recording Electronic voting system. Having such a model in mind, it can be easily explained why the various patches that are currently suggested (e.g. VVPAT or witness systems) are inappropriate.

Let's consider the following voting system (described by David Dill [HBO06]): a voter arrives at a polling place, and after proper identification, is given a ballot and proceeds to the voting booth. In the voting booth there is a human with a paper notebook and a pencil, siting behind a thick black curtain – call him *The Recorder*; through the curtain the voter tells *The Recorder* her favorite candidates and gets back a verbal assurance that her vote was cast.

Aside from the privacy problems (because the voice of the voter can be familiar to *The Recorder*), let's consider the problems that can arise with regards to integrity:

- *The Recorder* can write down a totally different vote than that communicated by the voter. This can happen because *The Recorder*:

  - did not hear the choice properly
  - heard the choice properly, but did not like it
  - is a supporter of a particular party
  - made a mistake (e.g. “misspelled” a name)

- The Recorder did not write anything at all;
- The Recorder wrote down some extra votes for the contests the voter did not vote.

At the end of the day, each *Recorder* counts the votes he recorded and gives totals to the poll workers. The notes of all the recorders along with their declared tally are then transported to the election headquarters and tallied. Results are declared. At this stage problems that can affect the integrity can arise:

- the count done by each *Recorder* is simply wrong
- once the *Recorder* sees the count, it decides to change all the ballots to favor a losing candidate
- the entity that transports the votes from the polling place to the central place
  - changes the ballots in transit;
  - loses some ballots;
  - alters some ballots;
  - injects some ballots;
  - disappears altogether;
- the counting at the election headquarters is wrong;

If the human Recorder is replaced by an electronic recorder, i.e. a computer, a Direct Record Electronic voting system is obtained, a DRE for short. All the attacks described above are still possible. The DRE can capture the vote incorrectly from the voter because of a miscalibrated touch screen or a poorly designed interface, can internally decide to flip votes from one candidate to another or an electrical surge can cause it to simply record the vote with errors or not record it at all.

DREs are black boxes; what happens inside them is extremely difficult to check. Claims can be made on what DREs are supposed to do, but independently checking that they worked properly is difficult. Proper testing before the election, federal and state certification, code inspection and extensive mock elections can detect many of its flaws, but not all of them. While it is possible to prove the presence of a bug, the absence of a bug is virtually impossible to prove. Because the probability that something can go wrong exists, it does not mean that it will go wrong, but it does mean that a valid question to ask is: can something better then a DRE be built?

Accessibility is an important plus of DREs. Their support for audio, large fonts and high contrast makes them usable by voters who are visually impaired or legally blind. Because a DRE is usually rather small, it can be installed in places that have wheelchair access or can even be transported to voters that cannot move. The immediate feedback to the voters can prevent accidental markings and reduce the number of residual votes. The inability to over-vote and the warning for under-voted contests are a valuable feedback that the voter gets without leaving the voting both.

Ballots can be presented to voters in a variety of languages, and the voter can switch between languages before starting or even during the voting process. For example a voter may start to vote on an English ballot but may switch to another language when a referendum question comes up and the text is long and contains words that are not familiar to the voter. The number of languages is virtually unlimited and the cost of offering them is negligible.

2.3 Software independence

As mentioned earlier, the main challenge of the voting problem is to prove, while maintaining the secrecy of the individual vote, that the tally is correct. There are two fundamentally distinct ways to approach this:

- To test the equipment several times before the election; if it performs flawlessly, to conclude that it will do so during the election.
- To produce a proof of correctness for each election, such that anyone may check the proof.

When using voting equipment, it would be ideal if it could be proven that the operation of the equipment is correct under all circumstances; the equipment could then be used over multiple elections. Unfortunately, voting systems are complex, having both complicated hardware and millions of lines of software code. If they are accessible to persons with disabilities, even more complexity (both hardware and software) is added. The best example of such a system is a Direct Record Electronic machine or DRE. It usually has a touch screen attached to allow voter input. Manufacturers build such systems from common hardware, commercial off the shelf software (COTS) and custom made computer programs. For a single election, a DRE can potentially serve hundreds of ballot layouts, and tens of different languages, in both visual and audio form. Commercial voting systems based on DREs use general purpose hardware and operating systems [FHF06], which are neither necessary nor desirable from the point of view of verification [Sal07].

It has been theoretically proven that there is no mechanism to prove that a general computer program is correct (the Entscheidungsproblem with proof given by Turing in 1936 [Tur36]). An intuitive argument is the existence of the halting problem: a program cannot be proved correct because it cannot be proved that it stops. While there may be a mechanism for proving correctness for a particular program or class of programs [Hoa69], in practice, in all but the trivial cases, given a program, it is extremely difficult to prove that it behaves correctly and does not have any collateral effect (e.g. because there may be an extremely large or infinite number of inputs or states).

Even though, in theory, a moderately complex program cannot be *proven* to behave correctly, in practice many software systems *do* behave correctly: online banking systems, airline ticket reservation systems, flight control software, computer games. Voting systems, have, however, failed many times and in many ways; as a result there is considerable doubt about their (general) reliability [Wikf, Gid05, Bin06, Mor04, New02, BM]. Researchers attempting to address the voting problem by means of securing the equipment have an extremely difficult job. Until they are successful, it is important to shift the paradigm: from one in which the voting software is trusted after several experiments, to one where voting software correctness does not play an important role in the verifiability of an election. If a bug or intentional change in the software alters the results of the election, it must be possible to observe that the tally is incorrect by looking at the results themselves, along with additional information provided by the voting system. Thus current research in voting system design focuses on the second approach, that of verifying the correctness of an individual election.

Rivest and Wack [RW] suggest the following definition.

Definition 11 (Software Independence (SI))**.** *"A voting system is software-independent* *if an undetected change or error in its software cannot cause an undetectable change or error in the election outcome."*

In other words, it doesn't matter what the election software does, the auditors will catch it if it misbehaves in an election. In this paradigm, the voting system may be viewed as a black box that produces a tally and a proof. The proof demonstrates that the tally is correct. The following are important points to note about the definition:

- The software of the voting system mentioned in the definition is the one used during the election process, supplied by the vendor, tested, audited and approved by the testing laboratories. It does not refer to software that was written by third parties who may be checking a property of the voting system, such as tally correctness. For example, any entity is free to write her own tallying software that would take data outputed by the voting systems and check the final tally, according to the tallying rules.
- The words "undetected" and "undetectable" were carefully chosen: an undetected change or error in the voting software refers to the past: the vendor did all the imaginable tests, the testing labs inspected the software and no error was detected. An undetectable error in the tally refers to the future, to the possibility of an error, one that ought to be discovered during the election or soon after it.

If the proof provided by the election software does not ensure 100% accuracy of the tally, software independence is referred to as virtual. In practice, a sufficiently, and ideally arbitrarily, large probability of correct behavior is acceptable (e.g. the probability that a number of votes no more than half the victory margin were not changed is greater than 99.999%). Another acceptable possibility is that the probability of cheating on n ballots decreases exponentially with n.

Having only a proof that the election software misbehaved may not be sufficient or even acceptable for election officials. Ideally, the detection of an error should be followed by its correction, without the necessity of rerunning the election (calling the voters back to the polls). Sometimes, correcting the error is possible and some new systems allow for retroactive correction (see Sect. 5.2.8). Some other systems are able to correct the error, but only if a privacy cost is paid (partial or total loss of ballot secrecy).

The definition of software independence articulates the paradigm shift: it is the election that should be verified, and not the equipment. The former is theoretically simpler, and has the benefit of being possible in practice. Verifying the equipment, on the other hand, is theoretically impossible, and there is substantial practical proof that failure of the equipment is common. If the tally verifies as being correct, it is irrelevant whether the software used to compute it contains bugs or not; the verification demonstrates that, for this particular election, the bugs did not interfere with the proper counting of the votes. It is possible that the conditions under which the bug actually does influence the tally are so unlikely that the bug, though it exists, will not ever affect the tally for the entire lifespan of the voting software. Thus, it is sufficient to require that each election be verifiable independent of the equipment.

Software independence does not mean procedural independence. It is often the case that software independent voting systems rely on procedural elements that may be easy or difficult to follow. Some have long and cumbersome procedures, others have a smaller number of steps to follow. In all cases, however, if the procedure is not followed, the integrity of the election is no longer guaranteed by the voting system. In other words, a shift is made from: "one must build it right and use it right", to "it doesn't matter if it is built right, as long as it is used right". Procedural compliance is, in most cases, a human process, thus the education of election officials is an important element in the paradigm of software independence. Officials need to be educated to understand what to do, and why it is done; finally they will be held accountable by the electorate.

2.4 End-to-end voting systems

Algorithmically speaking, voting systems perform multi-party computations, as defined in Sect. 2.1. At one end there are the true intentions of the voters; at the other end, the tally. Inspired by the end-to-end argument [SRC84], one can say that as long as the true intentions of the voters are all reflected in the tally, then the voting system is end-to-end verifiable.

Definition 12 (End-to-end verifiability (in voting systems))**.** *An end-to-end voting system can convince an independent observer that all the inputs are aggregated correctly.*

The word "ensure" in definition 12 does not necessary refer to 100% assurance; a probabilistic assurance may be sufficient. Also definition 12 does not refer to who is doing the checking or when the checking is done. Because the problem that it addresses is very hard, the easiest way to solve it is to break it down into three smaller properties, and to combine them into one uniform solution. Definition 13 does just that and it also specifies who and when the checks are done.

Definition 13 (Tallied-as-Cast verifiability (in voting systems))**.** *A voting system is said to provide Tallied-as-Cast verifiability if and only if it has the following three properties:*

Cast as intended *any voter can get a receipt after casting a ballot. Anyone should be able to check that all the receipts, with high probability, contain , respectively, all the true intentions of the voters.*

Recorded as cast *anybody holding a receipt that, with high probability, looks correct, must be convinced that her receipt is correctly recorded, at any time after getting the receipt.*

Tallied as recorded *anybody can convince himself that, with high probability, all the receipts are correctly transformed into the final tally, at any time after the final tally is audited.*

Definition 13 is more detailed and easier to check for a voting system. If the three properties, cast as intended, recorded as cast and tallied as recorded are put together, it results in tallies as intended, the property in definition 12.

The notion of end-to-end verifiability, as presented in this work, is not related to privacy. Note that a voting system that does not use a secret ballot does satisfy definition 13: if every voter posts her ballot publicly next to her name, all the three sub-properties can be easily verified. This means that end-to-end verifiability is one of the desired properties of the systems, but definitely not the only one (Sect. 2.1 contains a list of desired properties). In other literature (for example in the VVSG 2005 [Ele05]), some other properties have been included in the notion of end-to-end verifiability – for example, that end-to-end systems have to use cryptography, and need multiple trustees to decrypt receipts. This dissertation decouples these properties from the core of the definition.

Like software independence, the notion of end-to-end verifiability does not refer to the equipment used by the voting system. For each election, the verification steps from definition 13 must be checked, and thus each election gets its proof that the voting system worked correctly (or that it didn't) in this particular election, regardless of the equipment. The election is checked, not the equipment.

2.5 The relation between Software Independence and End-to-End

Definition 14 (Cryptographic Voting System). *A Voting System is said to be cryptographic if it uses any secure cryptographic function.*

ThreeBallot [Riv06] is the first attempt to create a voting system that is end-to-end verifiable and does not use cryptography. Unfortunately, the protocol is not software independent, as the integrity of the election relies on the correctness of the checker. VaV and Twin [RS07] have similar problems. For example, the security of Twin relies on the fact that the box that provides a copy of the ballot is secure.

The relation among end-to-end, software independent and cryptographic voting systems is pictured in Fig. 2.1. This is the most general relation, and it illustrates that end-to-end voting systems are not necessary software independent or cryptographic. The Venn diagram illustrates that software independence (as per definition 11) and end-to-end verifiability are not a sub or super set of each other. Examples of voting systems that have one or more of these three properties are depicted in Table 2.1. Ideally, a voting system is software independent and end-to-end but not cryptographic; this is an open problem since no such practical, scalable systems has been proposed so far.

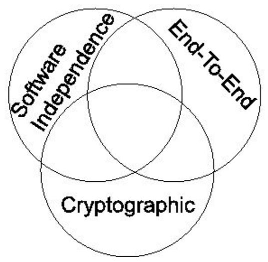


Figure 2.1: The relation between Software Independent, End-To-End and Cryptographic voting systems

The existence of ThreeBallot is one of the reasons for end-to-end verifiability and software independence being considered distinct properties by this dissertation. In ThreeBallot, a corrupt checker may allow a voter to cast an invalid ballot, for example one that has three votes for Alice and none for Bob (or a single vote for Alice and a negative vote for Bob). Let us consider that in an election there are 10 voters and 6 of them favor Alice and cast a valid ballot for Alice (two votes for Alice and one vote for Bob on each of the ballots), but the other four cast invalid ballots with three

	End-to-End	Software Independent	Crypto
DREs	No	No	No
DREs with encrypted EBIs[a]	No	No	Yes
Optical Scan	No	Yes	No
Optical Scan with encrypted EBIs	No	Yes	Yes
ThreeBallot	Yes	No	No
Three Ballot with encrypted EBIs	Yes	No	Yes
none currently	Yes	Yes	No
PunchScan, Pret-A-Voter	Yes	Yes	Yes

[a]Electronic Ballot Images

Table 2.1: Examples of voting systems and their properties.

votes for Bob and no votes for Alice. Once the $3 \times 10 = 30$ ballots are posted and the tally is performed, the total reveals that Alice got 12 votes and Bob got 18 votes and thus Bob wins conformably, in spite of the fact that a majority of voters wanted Alice to win. Thus, there is an undetectable error or fault in the tally and one of its causes is an error or fault in the voting system's software (the checker), thus the system is not SI. A simpler example is an election with a single voter casting, with the complicity of the checker, an invalid ballot; in this case it would be obvious that the tally (of the single vote) must be wrong[1].

2.6 Commitments

This dissertation considers a class of cryptographic voting systems, which use modern cryptographic tools to provide verifiability and privacy. From a technical point of view, the strength of a voting system is based on the soundness of the algorithms used and the hardness of the cryptographic algorithms that provide the privacy and verifiability. Central to the family of voting systems studied in this work is the notion of a cryptographic commitment. A formal definition of a commitment is given here.

Definition 15 (Commitment Scheme)**.** *A commitment scheme is a protocol that allows a prover to commit herself to a fact and prevents a verifier from learning the fact from the data resulting from the commitment.*

The four properties of a commitment scheme, as defined in [BCC88] are:

- The verifier can commit to a fact

[1]This situation may be viewed as being similar to the one in Volusia County [Wikf]

- Once committed, the verifier can open the commitment in only one way
- From the commitment itself, the verifier cannot learn anything about the committed fact
- A commitment scheme is not dependent on the fact committed to.

Commitment schemes are at the heart of zero knowledge proofs (ZKPs) [BCC88], which convince the verifier that an assertion is true, but do not allow the verifier to determine any more information beyond this fact.

The most common explanation of commitments is through physical analogies: Alice writes down a number at the bottom of a chest, locks the chest and gives it to Bob. At the revealing stage, Alice gives the key to the chest to Bob. Bob could not have learned the number while in possession of the chest, and Alice could not have changed it.

A commitment scheme can be perfectly binding or perfectly hiding [DN02]. Given a perfectly binding commitment scheme, the prover cannot change the value committed to, even with access to unbounded computational power. In a perfectly hiding scheme, the verifier would not be able to reduce the probability of the message after seeing the commitment, even with access to unbounded computational power. It is easy to show that both properties cannot hold simultaneously. Here is a brief sketch of the proof: if a commitment scheme is perfectly binding, then there is only one way to open the commitment; choosing a message at random and computing its commitment using the public commitment algorithm results in a commitment that is either equal or different to the initial one. If equal, the only message that fits the initial commitment has just been found, thus it is clearly not perfectly hiding. If not equal, then the probability distribution (or the density of the probability distribution) changes, since the guessed message has been eliminated; therefore the scheme is not perfectly hiding.

Chapter 3

Related Work

This chapter presents research in voting that is relevant to the contributions of the dissertation. A brief history of voting technology is presented first, followed by a description of voting technology most commonly used at this time. The chapter then focuses on the three fundamental cryptographic concepts in voting research—mixnets, homomorphic counters and blind signatures—and on examples of end-to-end-independently-verifiable voting systems.

3.1 Short history of election technology

We present a brief history of voting technology, based on [Sal06, Jon03, Bel]. One of the oldest documented forms of voting comes from Greece, in the form of an ostracon [Wikb]. All eligible voters (land owners) would gather in the central agora, where a porcelain vase was broken into pieces and each voter would take one piece and write on it. The votes were counted on the spot.

Starting with the first centuries A.D., the Pope was elected (as opposed to being appointed). In the 12th century A.D. the Third Lateran Council introduced the secret ballot for electing the new Pope, the first documented election with a secret ballot [Wikc].

In the early days of voting in the United States, elections were dominated by chaos, being conducted in public, most often viva voce (each voter says out loud the name of the desired candidate). Massive fraud was very common, especially by voter intimidation, bribery and ballot box stuffing (hence the expression "Vote early, vote often"). Paper ballots began to be widely used immediately following the American Revolution; South Carolina and the states of New England were already using them.

Tickets were commonly published in newspapers, and political parties encouraged voters to clip them out and bring them to the polls. Because tickets were of different shapes and colors, and one could typically identify which ticket a voter brought with him to the polls, the notion of the secret ballot was only realized in a limited fashion.

The Australian ballot was first used in 1856 in Victoria, Australia and soon afterwards in the south of Australia and New Zealand. In the US, the first legislation to enforce the Australian ballot was passed in 1888, in Louisville, Kentucky. According to [Sal06], Massachusetts in 1891, and, according to [Bel], New York in 1889, were the first states to adopt the Australian ballot for elections of national office.

At the end of the nineteenth century, taking advantage of the trend towards mechanization, the first voting machine was introduced by Jacob H. Myers. The first election conducted on a lever voting machine was conducted in 1892 in Lockport, New York. For the next one hundred years, lever machines were used all over the US, becoming the technology of choice in the fifties, with more than half the votes being cast using lever machines [Bel]. They are disappearing slowly (in 1996 20% of the votes were cast on lever machines), and in 2008 they are use in only about 5% of the jurisdictions. The technology was considered by many reliable and secure, and their disappearance is attributed to the lack of skilled individuals to maintain them, and to their general unavailability.

The next important step was the introduction of punch cards, used for the first time in voting in 1960, by two professors at U. C. Berkeley who improved the Port-a-Puncher pre-scored punch card system developed by IBM. Punch cards were never intended for wide unsupervised use, but, somehow, it seemed natural that anyone could use a stylus to remove a pre-perforated chad from a card to vote in a medium size election. Along with some other causes (see [RH]), the widespread use of punch cards by regular voters finally led to the technical disaster in the US Presidential Election in Florida in 2000.

Optical scan, or mark-sense, ballots were introduced in voting in 1962, twenty-five years after IBM introduced the technology for scoring standardized tests (which is still in use today). On an optical scan ballot, a voter fills in an oval (or connects two arrows) with a number two pencil. The ballot is then fed through a mark sense machine, which is usually not a regular digital scanner. The machine detects if pencil marks were made at certain geometrical positions, and associates them with a preset map to determine the candidate that corresponds to that geometric position. The technology is in wide use today, about 40% of the ballots in the US election of 2006 being cast on optical scan systems [Ser06]. Currently, it is becoming the technology of choice for many US states (e.g. California [Bow]), as the use of optical scan ballot systems allows manual recounts.

Computers have been used to count the punched cards or the mark sense ballots. While, to this day, election fraud using computers has not been proven in any election, a 1969 article in Los Angeles Times [Ber69] is perhaps the first to observe that it is possible. At that time personal computers were non-existent (HP's attempt to introduce one in 1968 failed, and microchips appeared only in 1971), however a red-team - black-team experiment was conducted in 1969 that showed that the result of the election could be rigged using the central tabulator, without leaving any evidence.

Before the problems with voting in Florida in 2000, the voting equipment used in many jurisdictions was believed to be working. No case of fraud using voting technology was ever proven, although researchers had drawn attention of the possibility of such fraud [Sal75]. Heavy media attention to the failure of voting technology in 2000 led to a change: it began to be seen that the US needed to develop more reliable voting technologies (technologies that would accurately capture voter intent, and accurately count votes). The objectives of election transparency—giving the voters the power to check their ballots, even after the election, and allowing anyone to check that the tally was correctly produced—were not a large concern expressed in the media reports, but drew attention from academic researchers and security experts.

3.2 DREs

Direct Record Electronic (DRE) technology can be traced back to 1974 [MZK+74]; and has its roots in a more general concept proposed in 1850 (Henderson) and 1869 (Edison). Currently, a DRE consist of a personal computer with low processing power (x386 processors are not uncommon) and little memory. A touch screen is used to allow for voter input. DREs are very flexible in terms of presenting a ballot to the voter: the fonts can be adjusted, multiple languages are supported, and audio ballots can be provided to voters with disabilities.

3.2.1 Problems with the DREs

Because DREs are nothing but computers, and computers have been used in elections for many decades now, problems have been known to exist for several years, as illustrated by the aforementioned Los Angeles Times article. But, even long before the existence of computers, when mechanized and automated voting was gaining ground, some reports raised concerns. See, for example the report of 1925 by David Zukerman [Zul25], which mentions that there exists a “lack of voter assurance that [the] vote will be counted in accordance with desires”.

Each US election since the year 2000 has been a good source of reports on DREs that fail during election day, see, for example: [New02], [BM], [Mor04], [Bin06], [Gid05], [Wikf]. The most common problems are miscalibrated touch screens, insensitivity to double touching, software freeze, inability to boot, printer jam, inability to count, non-zero initialization, full logs, out of permanent storage, and spontaneous reboots.

Various teams have analyzed the software and physical security aspects of DREs, some at the request of state officials, with the voting systems officially supplied by vendors. Other groups took advantage of improper handling of the code by the vendors. The Hopkins report of 2003 [KSRW03], SERVE [JRSW04], the RABA report [Cel04], the Caltech/MIT report [Pro01], the Princeton report [FHF06], all analyze specific vulnerabilities of a specific voting system and conclude that it is not safe to use in any election. Their recommendation is to equip the voting system with a voter verifiable paper trail, that would mitigate or eliminate many of the possible attacks. Sect. 3.2.2 discusses these recommendations and concludes that they are not appropriate for ensuring a secure and transparent election. Numerous other organizations are declared enemies of DREs alone, and many of them support a paper trail [Har, Ver, Vot, Bra].

At the beginning of 2007, Debra Bowen, the California Secretary of State ordered a top to bottom review of all four DRE like voting systems certified for use in the state. It was the first state-wide inspection ordered and paid for by the state, and investigators were given access to official machines, source code and documentation. Four teams were constituted and in the course of two months analyzed three of the four systems (one was never submitted for audit). The teams created a report that became public in August 2007 [Bow]. These voting systems were all found to have vulnerabilities that would make them insecure in public elections. As a result of the report, days after its release, the State of California decertified all four systems and reverted to state-wide precinct-based optical scan (one DRE per polling place was kept to be used only by voters with disabilities).

Thus there are two major facts that point towards the inappropriateness of the use of DREs as voting machines

1. DREs have failed to operate properly during real elections.

2. DREs cannot be checked independently for proper operation, thus a clever hacker could modify the software without leaving a trace, and could tamper with the results of the election.

These two facts point to two possible approaches: a careful and systematic design and analysis of voting machines, having reliability and security as a priority (the computer security approach); or the abandonment of DREs as a model, and the introduction of new voting systems that do not rely on the proper functioning of any voting software (the cryptography approach). Of course, a combination of the two approaches is also possible.

3.2.2 Independent Voter Verifiable Records and Voter Verifiable Paper Audit Trails

Many of the reports mentioned in Sect. 3.2.1 suggest that the introduction of a paper trail that the voter has access to would mitigate or even solve many of the inherent problems with DREs.

Rebecca Mercuri proposed the addition of Voter Verifiable Paper Audit Trails (VVPAT) to DREs [Mer00]. The trail is a piece of paper that is printed in the voting booth, in the presence of the voter. The voter is allowed to check that the paper record is consistent with her choices. If she wishes, she can also compare the paper trail with the information shown on the summary screen of the DRE. The paper is then deposited into an opaque ballot box, and used in a manual recount to check the correctness of the tally, and, thus, the validity of the electronic records. This can be mandated in a fixed percentage of the polling places, or can be dictated by a judge at the request of the candidate that lost. Having artefacts of the voting process that can be counted by hand is reassuring for the voters, and provides a natural fall-back mechanism.

In the first draft of the Voluntary Voting System Guidelines (VVSG) 2007 [Ele07] developed by the Technical Guidelines Development Committee (TGDC) appointed by the Election Assistance Commission (EAC), itself appointed as a consequence of the Help America Vote Act (HAVA), Independent Voter Verifiable Records (IVVRs) are mentioned as a generalization of VVPATs. The record may not necessarily be made out of paper (it can be plastic) or the record can be marked by the voter herself, as with an optical scan ballot, and then be read by a machine. Systems with IVVR are classified by the VVSG 2007 as being software independent, as errors in the voting system software can be determined through the audit trail.

The validity of a recount using an IVVR is only meaningful to those individuals doing the recount. If Alice gives Bob $10,000, Bob is not going to trust Alice, but is going to count the money himself. A hand count has meaning only for the one doing the count. If the voters or the losing candidates are not allowed to do a recount, they are forced to trust the one done by the election officials. Additionally, even if those performing the recount are deemed trustworthy, the integrity of the recount depends on the security of the chain of custody from the moment the IVVR leaves the hands of the voter until it reaches the hands of the person doing the recount.

3.2.3 Witness systems

This section discusses witness-based voting systems [Ger01, tea], sometimes called frogs [BJR01], and their conformance with the definition of software independence. The definition of software independence refers to voting software only, referring to the software that is officially used in the election to cast and count the votes, which was tested and certified by the national laboratories. Any other software that is used in any stage of the election process is not considered voting software and therefore the definition of software independence does not apply to it.

There are two clear cases: the witness system is either part of the voting system, or it is not. If the witness software is part of the voting system, then it is not a witness anymore, since it cannot be an independent witness of something that it is a part of. If the witness is not part of the voting system, there are still some outstanding major problems with the use of witnesses. We present only a small fraction of the problems here. Our discussion assumes that the witness is a programmable device; for example digital cameras do have software in them, therefore they can be reprogrammed to execute software that serves a particular cause.

The main problem is the authenticity of the records of the witness system. How does one know that the witness did not simply invent some records? For this reason, one witness is not enough. Having two or more witnesses is very impractical and gives rise to questions of which witness is more credible. The problem has been falsely compared[Ger01] to the notion of agreement in distributed systems; since the witness systems do not scale as distributed systems do, the analogy is misleading.

A resolution procedure for disputes between the witness system and the voting system would be very complicated. Since the witness is not part of the voting system, and it has not been tested by the laboratories, for example, the incorrect functioning of tested and trusted voting system must be asserted by an untrusted and untested witness. This is of course, absurd, since it might have been that the witness did not function correctly.

Another problem with the use of witness systems is that the privacy of the vote can be adversely affected. The witness may have access to the order in which the votes are cast on a voting system. Since the witness is not audited by anyone, there is not even an indication that the order is scrambled. Also, identifying elements can be actively inserted by voters, such that anyone with access to the witness can connect a vote with a voter. In the case of a photo camera, the voter can interactively get into the picture, e.g. as instructed by the coercer.

Witness systems are explicitly mentioned in the VVSG 2005 [Ele05], as an independent verification system. A witness system is not conforming to the definition of SI because an undetected error or fault in the voting system software can produce an undetectable error in the tally, since the witness system cannot provide convincible proof that the tally is incorrect. Therefore the non-conformity is not because the witness systems is part of the voting software or not, but because the tally is not error free.

3.3 Capturing the intent of the voter

While this dissertation does not contribute to research on the distinct ways of aggregating voter intentions, we present a summary here for completeness; there are several sources for details, see, for example, [SK].

In a typical election, a voter is presented with a set of choices on one question, or office, and asked to choose one. The single winner voting system, also called first-past-the-post or plurality voting, obtains the least information from the voter, who provides a "yes" vote for a single choice. If more than one question or office is available, then the voter may be allowed to choose one choice for each, which corresponds to many elections running concurrently. The winner for each question/office is the choice with the greatest number of votes. Plurality voting can lead to the *spoiler effect* [SK07] when there are more than two choices; a vote for the candidates least likely to win takes votes away from the main candidates, thus voters who prefer the candidates least likely to win have to choose between stating their true preferences and casting a vote that may make a difference.

A generalization of plurality voting is to allow the voter to make more marks than the number of offices. This is called approval voting [BF78] and the voter is allowed to vote for as many or as few candidates as she wishes. Again the winner for each question/office is the choice with the greatest number of votes. This allows the voter to communicate her opinion about all the candidates, indicating that some are preferred and others are not. However, from among the preferred ones, it fails to indicate the order of preference. This is remediable in preferential voting [Wikd] (also called rank voting), where the voter is allowed to order the candidates according to her preference. Declaring a winner in a rank voting contest is more complicated than simple summation. First, there is no single method for aggregating the preferences, and different methods may declare different winners. The most common methods are: single transferable vote or STV (also called instant runoff voting, IRV), Borda-Count and the Condorcent method; they differ in how the winner is declared, but all allow to specify the preference of any candidate over any other candidate. Some examples where rank voting is used are: Australia, Ireland, The United Kingdom and Cambridge, Massachusetts [Wika].

Range voting [SK] goes one step further than rank voting and allows the voter to compare candidates both qualitatively and quantitatively. In range voting, the voter can assign a score (usually between 0 and 9) to each candidate. A higher score indicates a greater preference. Scores for each candidate are averaged and the one with the highest average is declared the winner. Rank voting is currently used in the Olympics (e.g. artistic ice skating). There are studies that show that bees choose their next nest using range voting [Smi07].

Write-in candidates are allowed in most jurisdictions in the US. Their mere existence is seen as an extension of democracy, because if a candidate is not allowed to enter the race by bureaucracy, the voice of the people can still prevail. A contest can have a special "write-in" option where the voter may write down the name of her favorite candidate. Write-ins are used in the US only, and have rarely led to electing anyone to an office. (Anthony A. Williams, the rare exception, won the mayor seat in Washington DC as a write-in candidate in 2002.) Write-ins are difficult to count without human intervention (except, perhaps, with DREs).

A good voting system is able to support any of these methods. It should support both the capture of voter intent, as well as the determination of the election outcome, using any of the methods.

3.4 Cryptographic Voting Systems

There are three classical cryptographic techniques for electronic voting:

- Homomorphic counters
- Blind signatures
- Mixnets

The following sections present the general techniques for each of them, along with examples of some of the voting systems that use them.

3.4.1 Homomorphic counters

In a homomorphic voting system, the cleartext ballots are never visible to anyone except the voter. The encrypted ballots are made public and are aggregated in encrypted form. The encrypted tally is then decrypted. These systems require special types of *homomorphic* encryption schemes, and homomorphic counters, which enable the computation of the encrypted tally from the encrypted votes. A function F is said to be an $(\oplus, \otimes)$ homomorphism if $F(a) \oplus F(b) = F(a \otimes b)$. In particular, if F is an encryption function, and a and b are votes, and $\otimes$ is regular addition; the encrypted tally is obtained by applying $\oplus$ to the encrypted votes. In a homomorphic encryption scheme anyone can check that the encrypted tally is computed correctly, as all the encrypted ballots are public. A physical analogy is the lever voting machine: every time a voter pulls a lever, a counter is incremented; at the end of the day the counters are made public. Using homomorphic counters, the voter has the additional option of verifying the proper recording of the vote anytime after casting it, as well as of checking that the homomorphic counter is correctly composed.

Homomorphic counters were first proposed for use in voting protocols by Josh Benaloh [Ben87, BY86]. To distribute the authority that is allowed to decrypt the counters, Benaloh used a classical secret sharing technique [Sha79]. Each of a small number of parties gets a share of each encrypted counter, it then decrypts the share, and the secret sharing technique is used to compute the cleartext value of each counter.

Because homomorphic voting systems do not produce cleartext ballots, it is difficult to adapt them to tallying rules other than regular addition. In particular, rank voting (with its various counting rules) is not consistent with the framework of a homomorphic system.

A particular challenge is to prove that the encrypted vote $F(a)$ is indeed an encryption of a valid vote. For example, in a race with two candidates and with two possible votes, -1 and 1, the entity that prepared the ballot (e.g. the voter) has to prove that the encrypted ballot contains either 1 or -1, but cannot contain some other value. If it would contain a 2 or a -2, then this would mean a double vote. This proof is the most costly operation in homomorphic systems, and is difficult to generalize to a larger ballot. One may use one counter per candidate; a vote would be an encryption of zeros and ones. This implies, however, the need for as many well formed proofs as candidates – which is very costly for even moderate size elections.

Following Belanoh's scheme [BY86] improvements have been made to address the verifiability of the secret sharing [Sch99], receipt-freeness[BT94], decreasing the number of rounds in communication[Sch99], efficiency and generality[AR06]. Notable voting systems using homomorphic counters are [BT94], [CGS97], [Sch99], and [LK00]. Recently, Adida proposed a simplified model for elections using paper ballots [AR06].

3.4.2 Blind signatures

A blind signature is a very general mechanism invented by David Chaum [Cha82] that allows a client to obtain a signature (on a message) from an authority, while ensuring that the signing authority gains no information about the message it signs. Blind signatures were invented to protect privacy in the context of electronic payments, using protocols very similar to voting protocols. Chaum anticipated the use of blind signatures in voting and mentions voting as an application in the original paper [Cha82].

Blind signatures require

- a public signature function (e.g. RSA), such that only the authorized party can produce a signature $Sign(x)$, and everyone can check that the signature is valid;
- a function $Blind$ and its inverse $Unblind$, that is commutative with $Sign$; in other words $Unblind(Sign(Blind(x))) = Sign(x)$. When $Blind$ is applied on x, $Blind(x)$ does not communicate any information about x.

There are many flavors of voting systems that use blind signatures; we explain the basic principle. The assumptions are that a blind signature scheme and an anonymous communication channel are available. The steps are as follows:

- before election day:
  - the voter fills in a ballot and attaches to it a randomly chosen (long) serial number; denote the ballot with attached serial number by x;
  - the voter blinds x, by applying $Blind(x)$;
  - the voter digitally signs $Blind(x)$, with her personal official private key and sends the signed $Blind(x)$ to the election authority;
  - the election authority checks the signature and checks that the voter has not already voted in the current election;
  - the election authority signs $Blind(x)$ to obtain $Sign(Blind(x))$ and sends it back to the voter;
  - the voter applies $UnBlind$ to get $Sign(x)$, $Unblind(Sign(Blind(x))) = Sign(x)$;
  - the voter checks that the signature of the election authority on x is valid;
- during election day
  - using an anonymous channel, the voter sends $Sign(x)$ to the ballot box;
  - the ballot box checks that the signature of the election authority is valid;
  - the ballot box checks that x was not cast before, by checking the serial number of the ballot (to prevent a voter from sending $Sign(x)$ twice);
  - the ballot box counts x;

Voting systems based on blind signatures are more suitable for remote voting, since the interaction between the voter and the centralized election systems is performed at different times. If the interactions were performed close together in time, then the election authority, in collusion with the ballot box, can determine the correspondence between signed votes and blinded votes, thus undermining privacy. The standard problems with remote voting systems (improper influence and the fact that the voter's machine knows how the voter voted) are not addressed by the blind signature based systems.

The voter does not get a receipt in the basic protocol. Hence, while the voter can check at any time that her ballot is correctly posted, she cannot complain if the posting is not correct and provide a valid proof, while protecting her privacy. The proof, if her vote is not posted, is a plain text signed ballot that would destroy her privacy. Additionally, the basic blind signature protocol does not protect against a coercer requiring the voter to use a certain serial number or a certain blinding factor, which would open the door to coercion.

A voting system based on blind signatures provides unconditional privacy, as the blinded ballot does not reveal information on the ballot itself. On the other hand, the integrity of such a voting system is guaranteed only if the adversary is computationally bounded; an unbounded adversary could break the signing algorithm and stuff the ballot box.

After blind signatures were initially proposed [Cha82] (with voting in mind), Chaum proposed the first complete voting protocol [Cha88b]. It was followed by numerous schemes that addressed universal rather the individual verifiability[Rad95], efficiency[JL97], and number of rounds of communication [FOO93].

3.4.3 Mixnets

A mixnet follows most naturally from the traditional paper system: it allows the voters to cast hidden (encrypted) votes which are then all shuffled and revealed (decrypted).

Mixnets were first introduced by David Chaum [Cha81] as a method for anonymizing messages in general, and electronic email in particular. Chaum mentions, in the original paper, how a mixnet can be used for elections; this concept is retained today in mixnet-based voting systems. The voter prepares a vote privately (whether in a voting booth or not). The vote is then encrypted with M different public keys, one for each mix in the mixnet. The order in which mixes process votes is the reverse of the order in which the keys were used, so that the first mix is the one whose key was last used for encryption. The voter then authenticates herself to a public bulletin board and posts the encrypted ballot on it.

A mixnet is composed of a set of mixes, linked serially, such that the output of one mix becomes the input of the next mix. The very last mix outputs cleartext ballots. The first mixnet reads in encrypted ballots from the bulletin board, removes the first level of encryption using its private key, shuffles all the processed ballots, and makes them available—to the next mix and to the public—by publishing them on the SPBB. Thereafter, each mix partially decrypts all the ballots and shuffles them. Because the shuffling is kept secret by each mix, a ballot cannot be followed from the input of the first mix to the output of the last mix. Fig. 3.1 presents an overview of the process.

If the encryption used to encrypt the messages (votes) sent to the mixnet is deterministic, it is possible for an adversary to encrypt each output message with the

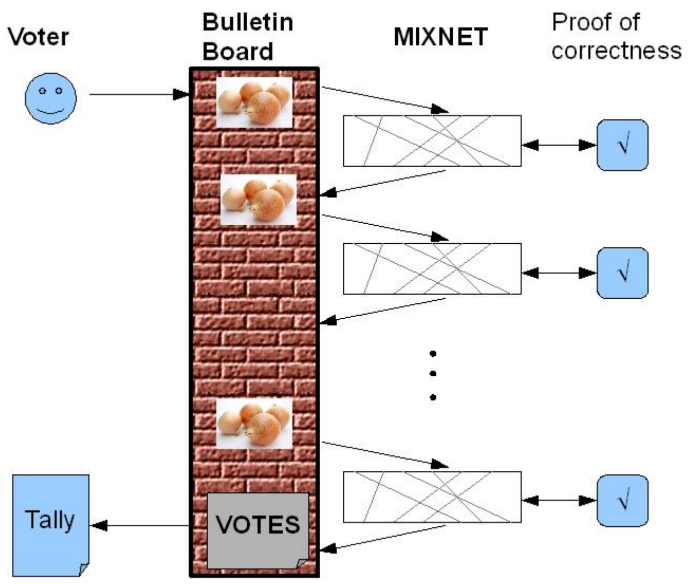


Figure 3.1: A general voting system using a decryption mixnet and a bulletin board.

public key of the mix to determine the corresponding input message. Hence, random numbers are used during the encryption.

More formally, Chaum assumes the existence of a public key cryptosystem, and a public/private key pair $K/\hat{K}$, such that, for any message X, $\{\{X\}_K\}_{\hat{K}} = \{\{X\}_{\hat{K}}\}_K = X$, where $\{X\}_K$ denotes the encryption of X using key K. For a single mix and a single message, W, the input is $\{R_1, W\}_{K_1}$ and the output is W, where W is obtained by decrypting the input $\{\{R_1, W\}_{K_1}\}_{\hat{K}}$ and removing the random value R_1. The output W can itself be of the same form, $W = \{R_2, W_2\}_{K_2}$, so many mixes can be cascaded. The input can be constructed such that the messages have to go though a number of mixes before finally getting the cleartext message. In this case, the output messages are of the form: $\{R_M, \{R_{M-1}...\{R_2, \{R_1, \{R_0, W\}_{K_0}\}_{K_1}\}_{K_2}...\}_{K_{M-1}}\}_{K_M}$. This particular type of mixnet is now called an onion decryption mixnet, because each mix removes a layer from the input by decrypting it before, mixing the output.

In practice, a hybrid encryption scheme, like PKCS#5 enveloping, is applied, where the payload is encrypted using a secret symmetric key *skey* and *skey* is encrypted with the public key of the mix. This is necessary because the payload grows with each mix, and the asymmetric encryption schemes cannot handle messages or arbitrarily large lengths. This has been observed in [PIK94] and an alternative solution was proposed, namely re-encryption mixnets. Instead of decryption, each mix performs a re-encryption of the messages. At the end, a single decryption recovers the messages. An encryption scheme that lends itself to this approach is the El Gamal scheme [Gam85], since a cipher text can be re-encrypted without knowledge of the private key. Park et al. [PIK94]also suggested that the private El Gamal key be shared, and each mix perform both a re-encryption and a partial decryption.

3.4.3.1 Correctness

So that coercers may not require voters to provide these as proof of how they voted, the random numbers are not chosen by voters, nor are they made available to voters after the vote is cast. The voter may ensure that her ballot was prepared and encrypted properly through the use of a cut and choose process: the voter challenges the encryption software and gets to see a chosen portion of the encryption process. For example, the software prepares two identical filled-in ballots, but encrypts each with a different set of random numbers. The voter then chooses one of the two encrypted ballots to open; the software reveals the random numbers used for that encryption. The voter checks that the encryption was performed correctly, and spoils the checked ballot, casting the other one (for which the random numbers were not revealed). Each voter has a 50/50 chance of catching a piece of software that cheats, the chance that cheating software will be caught rises exponentially with the number of ballots cheated on.

A provable mix is able to provide a correctness proof, of decryption correctness and shuffling correctness. We focus on decryption correctness first. Since mixnets were proposed in 1981, there have been many techniques developed that demonstrate that the mix did not simply inject or substitute some of the inputs. The two main techniques are: zero-knowledge proofs (ZKP) [GMW91] of equivalence, and randomized partial checking (RPC [JJR02]). A ZKP proves that the batch input to the mix decrypts to the output batch, without revealing information on the votes. The proof is usually complex and costly in terms of time, memory and number of steps. RPC consists of randomized checks of the links between the input and output values of the mix; it typically does reveal information. An auditor chooses certain inputs (or outputs) at random. The mix reveals the information necessary for checking that the requested inputs go to certain (different) outputs. For example, the mix reveals the random numbers that were used when constructing the encrypted votes with its public key, along with an indication of where in the output batch the decryption result is located. The auditor can simply take the output and the random numbers and re-encrypt them with the public key to check correctness. This is much more efficient than ZKPs.

We now focus on shuffle correctness. A very simple solution was proposed by Chaum in the original paper on mixnets: a mixnet would shuffle the output ciphertexts by simply sorting them. Correct shuffling may be trivially checked by checking that the output is sorted. Other methods for checking that a true random shuffle was performed were proposed in [Adi06] (but none as elegant as sorting).

3.4.3.2 Properties

A mixnet is a distributed authority; each mix (or a set of two to four mixes) is under the control of a different entity. Collusion of all entities can lead to the reconstruction of the full path from an encrypted input to a cleartext output. However, if a single entity is honest: correctly performs the encryptions and the random shuffle, and does not collude with the other entities, it is not possible to link the inputs of the mixnet with the corresponding outputs.

The mixnet does not participate in the construction of its input; it only publishes the public keys of each mix. If a single entity creates the mixnet input, and if the inputs are unique, the entity is capable of tracing the message though the entire mixnet. This is because each mix transforms each input in a deterministic fashion, and the entity that created the input will know how each message will look in each stage of the transformation. It is hence desired that a truly distributed entity creates the mixnet inputs. Depending on the encryption scheme used, this may (ElGamal) or may not (RSA) be possible.

The paths through a mix are dynamic, and can be chosen at run time by the mix. An immediate advantage is that an input batch can be processed multiple times by the same mix, yielding differently-ordered outputs each time, without having to make a special setup ahead of time. If a certain rule is used to create the permutation, like ordering the outputs lexicographically, then the paths are not dynamic anymore in a decryption mixnet.

The format of the ballot is not related to the parameters of the mixnet (the public keys), so ballot format can be chosen after the other parameters have been established. For an election, this implies that the list of candidates can be finalized late in the election year.

A mixnet is rather inefficient, mainly because of it uses of public key cryptography. Encrypting and decrypting messages using a public-private key pair often requires the computation of time consuming modular exponentiations. Sect. 5.4.1 contains a detailed performance analysis.

3.4.3.3 Mixnet Variations

A large variety of mixnets have been proposed, a short taxonomy being presented here:

- decryption mixnets;
- re-encryption mixnets;
- mixnets that decrypt and shuffle at the same time;
- mixnets that first shuffle and then decrypt (a shuffling mixnet first and a pure decryption mixnet second);
- mixnets using different encryption scheme;
- verifiable mixnets
- efficient mixnets

A good summary of the evolution of the literature on mixnets can be found in [Adi06]. For voting, many of the generally desirable properties of a mixnet are not needed at all, or are needed only in part. For example, many of the cast votes are the same, thus mixnets that perfectly shuffle messages that are completely different may be too general.

The 1990's witnessed considerable research on mixnets, with variants addressing correctness, verifiability and efficiency. Some notable work is [Cha88a], [MH96], [PP90], [OKST97], [ABE25]. In the early 2000's, efficiency became a concern, with efficient solutions proposed in [JJ99], [JJR02], [FS01], [Nef01b]. Voting systems based on the various ideas have been proposed at the same time, often the improvements to mixnets were motivated by the voting problem.

3.5 Remote voting

Advances in communication technology have reached a level where voters view remote voting as a natural evolution of polling place voting. It can be viewed as voting without the privacy of a booth; it thus inherits all the potential problems of polling place voting, but, in addition creates new challenges: how to ensure that the blank ballots correctly make it to the voters, and the filled-in ballots make it back correctly to some central location; how to ensure that the voter can express her free opinion without being influenced by surrounding elements, such as through coercion or bribery. While the delivery and return of ballots can be addressed through the implementation of strict procedural steps, the problems of coercion and bribery are more challenging, and are together referred to as the improper influence problem.

Internet voting is a particular case of remote voting, where the voter fills in a ballot using a computer and casts it via the Internet. The most challenging case is when the voter can use any terminal connected to the Internet to fill in and cast a ballot. Malware installed on the terminal can learn or alter the voters choices and still have the voter believe that her intention was kept secret and correctly transmitted. The transmission link can be listened to, but the attack on the transmission medium can be addressed by classical encryption techniques. Denial of service is a threat to any service offered via the Internet in general, and to voting in particular, especially if the voting period is short. Research in Internet voting systems has attempted to keep the same level of assurance as offered by polling place voting mechanisms, and to address both improper influence and the malware problem. We briefly describe some of the effort made by others in this field.

As early as 2001, a voting system called SVIS[Sak07], by Kazue Sako, based on [FS01] and [SK94], was used to run several elections over the Internet. SVIS collects encrypted votes and uses efficient mixing to produce the final tally; in some cases only the winners will be announced. However, the system addresses neither the improper influence problem, nor the malware problem.

Helios [Adi08] was proposed by Adida as a web-based voting system, essentially a web implementation of the Benaloh challenge voting system [Ben08]. It is very easy to use, fast and open source, but does not claim to solve the improper influence or the malware problems. Adder [KKW06], developed at University of Connecticut, has properties similar to Helios. The major difference between the two is that in Helios the election authority encrypts the vote, while in Adder the voter created the encrypted vote.

Civitas [MRCM08] was developed at Cornell by Clarkson et al. as an enhancement of the system proposed by Juels, Catalano and Jakobsson that uses both reencryption mixnets and homomorphic encryption Civitas is the first implemented system to offer protection against improper influence; however, the system is difficult to use, the tabulation and verification are slow, and it does not offer protection against malware.

"The Voting Ducks" [oT07] and Joaquim et al. [JR07] use the concept of a second channel, independent of the computer, that transmits to the voter codes for the candidates (unique per ballot). The voter would type in the code on the computer.

Both systems address both the improper influence and the malware problem, but suffer from usability and performance problems. Paul et al. [PER03] describe an authentication method using visual cryptography that can be used for casting a ballot on a potential infected computer, by having the ballot constructed as a large image, with the candidates spread across the image. While these techniques would prevent the computer from ever seeing the vote, all use a channel that is independent of the computer (e.g. sms or postal mail); thus their practicality is debatable.

3.6 A Survey of Mixnet-Based Voter-Verifiable Voting Systems

Many of the voting systems proposed in the past [Ben87, BY86, CGS97, Cha82, FOO93, Cha81] require that the voter has access to trusted computation to cast a ballot and do not allow the voters to complain if the receipt that is made public is not identical with the receipt they posses. provide a voter a receipt that the voter herself can check and use in case of a resolution dispute. The system may simply refuse to give a receipt, or give a receipt that is for a different ballot, etc. Only recently there appeared voting systems that solve these problems by allowing the voter to obtain a receipt that is:

- Always Available: the system always gives the voter a receipt and does not know if the voter kept the receipt or not
- Sufficient: no extra information beyond that on the receipt is required for dispute resolution

Always Available: The security of the receipt-based systems requires that (at least) a random (unpredictable) subset of voters keep their receipts and verify it. Hence, the receipt is available to all voters, and the voting system should not be able to differentiate between voters that do keep the receipt and those that do not.

Sufficient: Receipts are typically made public on a SPBB and the voters are free to check that their receipts are correctly posted. If a discrepancy is noticed between receipts and posted records, the receipt is sufficient to file a dispute resolution complaint. In some cases, the receipt itself is sufficient evidence of an error [PH06, CRS05]. In other cases, the receipt is only an indication that there might be an error [CCC$^+$08]. In such cases the system can eventually determine and prove whether there was an error or not.

This section describes in considerable detail the main mixnet-based voter-verifiable systems: CVV, Prêt à Voter and PunchScan; these systems are precinct-based and paper-based. A voting booth is provided to voters to allow them to select their candidates privately. The official ballot is made out of paper (or a similar write-once medium), as opposed to a strictly electronic ballot that the voter cannot check. PunchScan and Prêt à Voter are based on optical scan (but not mark sense), while CVV is based on an electronic machine attached to a specialized printer that produces the ballot to be cast.

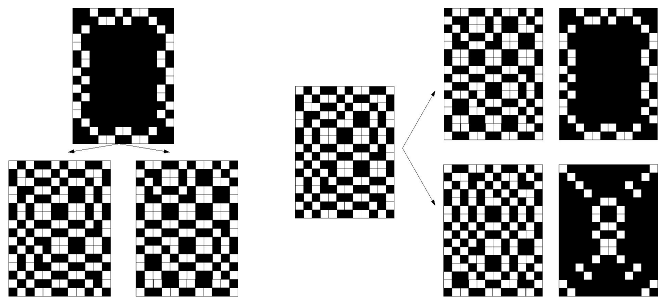
3.6.1 Visual Cryptography (CVV)

CVV[1] was the first voting system that allowed the voter to take home a privacy-preserving receipt and to later use that receipt to ensure that the vote was included in the final tally. The system was first proposed in 2004 by Chaum [Cha04]. The first detailed description was provided by Vora [Vor05]. It was implemented by a team from The George Washington University in 2004. Version 2.0 of the software was written by the author of this dissertation in 2005.

The voting machine consists of: a touch screen interface that the voter can use to make the desired selections, and a special printer. After the voter selects the candidate(s) and checks and approves the summary screen, the printer prints two transparent layers at once. The voter checks that, when overlaid, the two layers represent her vote(s). The ballot layers are constructed using visual cryptography [NS95]. Fig. 3.2 shows an example of a ballot in an election with two candidates: X and O. Any black and white image that has enough redundancy can be used by such a system. For example it can be an image containing the names of the candidates. The names can be chosen by the voter from a list of candidates, or can be typed in when casting a write-in ballot.

The voter chooses at random one of the layers to keep as a receipt and watches the other layer be destroyed. The printer prints additional information on the surviving layer, which enables the voter to later check, using software independent of the voting system, that the chosen layer is well-formed, and, if the destroyed layer had been the one chosen, it would have been correctly decrypted (notice the similarity with the traditional cut-and-choose proofs of correctness of ballot encryption). After election day, any voter can go to the web site (the SPBB) of the election authority, enter

[1]CVV [Vor05] is the name given by the GW team implementing a system invented by David Chaum [Cha04].


(a) A ballot containing a voter for ZERO. When the two layers are overlaid, the symbol ZERO is visible, but when looking at any single layer, no information is revealed.

(b) One layer can represent with equal probability a vote for either ZERO or X, depending what the other layer is

Figure 3.2: Sample ballot using Visual Cryptography

the serial number for her ballot, and check that her receipt is correctly retained on the web site. The receipt is decrypted using the traditional onion mixnet, which can also provide correctness proofs. For efficiency reasons, CVV uses a variation of the original randomized partial checks.

3.6.1.1 CVV Details

CVV uses an onion (layered) mixnet in the traditional manner described in Sect. 3.4.3. The mixnet is operated by trustees, each trustee maintains one or more mixes. Each mix publishes a pair of keys $(K_i, \hat{K_i})$, where K is the public key and $\hat{K}$ is the private key. For simplicity we assume that there is only one voting machine, having three pairs of keys $(K_V, \hat{K_V})$, $(K_V^t, \hat{K_V^t})$ and $(K_V^b, \hat{K_V^b})$, where $(K_V, \hat{K_V})$ is the key pair of the voting machine used for signing the receipt, K_V^t is a key pair associated with the top layer, and K_V^b with the bottom layer.

The ballot printed after the voter finishes her selections consists of two encrypted layers – $L_t(s)$ representing the image on the top layer for serial number s, and $L_b(s)$, the image on the bottom. When superimposed, these images produce the vote, $V(s) = L_t(s) \oplus L_b(s)$, where $\oplus$ represents the exclusive or function. Notice that each pixel (we will refer to this as a super-pixel) in $L_t(s)$ or $L_b(s)$ is represented by four printed pixels (we will refer to these as sub-pixels). The association of a group of sub-pixels with a particular super-pixel is such that, when two super-pixels are overlaid, the resulting super-pixel is the exclusive or of the two original super-pixels.

Each layer, $L_t(s)$ and $L_b(s)$, contains alternate pseudo-random and encrypted super-pixels (each super-pixel is a bit): one pseudo-random bit, one encrypted bit, one pseudo-random bit, one encrypted bit. The pseudo-random pixels are staggered one unit in the other layer of the receipt. Let l represent the layer, top or bottom, $l \in \{t, b\}$. $W_l(s)$ represents the pseudo-random bits of layer l, half the bits in the image $L_l(s)$. Its size is half that of the ballot image. The pseudo-random bits are generated by adding contributions from all trustees – each trustee's contribution is generated by a distinct seed, and the seed is generated by the polling machine using the serial number of the receipt.

Let h be a one way function. For the ballot with serial number s, the i^{th} trustee's seed for layer l is:

$$seed_i = h(i, \{s\}_{\hat{K_V^l}}) \tag{3.1}$$

and the pseudo-random string used in layer l is:

$$W_l(s) = \sum_i^N h'(seed_i)) \tag{3.2}$$

where h' is a secure expander (it deterministically produces a fixed number of bits from a given seed).

Printed at the bottom of both layers are three strings, identical on both layers: the serial number of the receipt and two strings representing the commitments by the polling machine to the seeds of the pseudo-random strings in the two layers. These strings are referred to as *dolls* (like Russian dolls, the general string contains other strings within it) in [Cha04]. They contain, in serial encrypted form, as in mixnets, the seeds required by the trustees to generate their shares of the pseudo-random bits on each of the two layers.

$$\mathcal{D}_l^i = \{seed_i; \mathcal{D}_l^{i-1}\}_{K_i} \tag{3.3}$$

where $\mathcal{D}_l^0 = NULL$ and the ballot has, printed on it, $\mathcal{D}_t^M$ and $\mathcal{D}_b^M$.

The voter now chooses one of the layers. In response, the machine now prints, only on the chosen layer, $\{s\}_{\hat{K_V^l}}$, which can be used to check the proper formation of all the seeds used to construct the pseudo-random bits on the chosen layer. The voter can walk away with the chosen layer, and later check that (a) the pseudo-random bits on her chosen layer were indeed generated using the seed – see Eq. 3.4, (b) that the seed itself is correctly generated from the serial number – see Eq. 3.4, and (c) that the doll that would have been used to decrypt the non-chosen layer is correctly formed. However, the voter cannot herself decrypt the receipt to prove how she voted, because the layer she possesses, l, contains the vote encrypted using the pseudo-random bits she did not open: $W_{\bar{l}}(s)$, generated using seeds $h(i, \{s\}_{\hat{K_V^{\bar{l}}}})$, where $\bar{l}$ represents the complementary layer. Further, these seeds are hidden in dolls that cannot be opened. The receipt hence does not violate the requirement that the voting system provides involuntary privacy.

All the information on all the voter receipts is posted on the SPBB. A voter can check the SPBB to see if her vote is included in the public collection. Further, she can also check that the polling machine did not cheat in the construction of her encrypted receipt. This is done by checking that the seed provided by the polling machine was used to generate the pseudo-random bits on the layer she possesses. The pseudo-random bits should have been generated by the seed, which itself should be a signature on the serial number. That is, if $y = \{s\}_{\hat{K_V^l}}$ is the string printed by the machine after the voter made a choice of layer, the voter may check if:

$$W_l(s) = \sum_i^N h'(h(i, y)) \quad \text{and if} \quad \{y\}_{K_V^l} = s \tag{3.4}$$

i.e. if $\{s\}_{\hat{K_V^l}}$ is the signature of s with private key $\hat{K_V^l}$.

Further, the doll corresponding to the layer the voter possesses can also be checked, i.e. it can be checked that the polling machine was correctly communicating, to the trustees, the bits used for encrypting the layer not chosen. If the string corresponding to $\mathcal{D}_l^M$ is represented by z, it can be checked by constructing all the dolls using the string y:

$$\mathcal{D}_l^i = \{h(i, y); \mathcal{D}_l^{i-1}\}_{K_i}$$

and that the final doll is indeed z:

$$z = \mathcal{D}_l^M \tag{3.5}$$

A voter may not want to be bothered by the tedious checking of an encrypted ballot and can instead hand it over to an organization she trusts. For example, the League of Women Voters can collect such ballots and perform the task of checking. Once checking is performed to everyone's satisfaction, the receipts on the public website are declared uncontested and are then run though the mixnet. Note that the receipts are digitally signed by the polling machine so that a voter who maliciously changes the receipt and makes a false claim will be detected.

Once the collection of receipts is declared correct, the trustees, running a mixnet, decrypt the votes. Once the trustees have finished processing the encrypted ballots, the vote totals are made public and the provisional election results are declared. If L_i is an output ballot image of the i^{th} mix (and the input to the $(i+1)^{th}$ one), the i^{th} mix computes L_i by decrypting the corresponding doll passed on by the previous mix, and using the information in the doll to add a contribution to the decryption of the corresponding encrypted ballot image. The result of the decryption has two parts. The first part is used by the mix for decryption of the input image. The second part forms the next trustee's doll.

$$\{\mathcal{D}_l^{M-i}\}_{\hat{K_i}} = (seed_i, \mathcal{D}_l^{M-(i-1)}) \tag{3.6}$$

$$L_i = L_{i-1} \oplus h'(seed_i) \tag{3.7}$$

where $L_0 = L_l(s)$, the image from the receipt and $\mathcal{D}_l^M$ is the doll printed on the receipt.

The last phase involves auditing the trustees. A random half of the messages going into the first mix are opened and, for these votes, a correspondence between the input and output of the mix is established. For the next mix, the other (complementary) half of the votes is opened. Thus, at the end of the first two mixes, after the audit, a single output message can be traced back to exactly half of the input votes. To improve on this, four mixes can be grouped together. For the third mix, half of the votes opened by the second mix, and half of those opened by the first mix, are opened. The fourth mix opens those not opened by the third one.

When the i^{th} mix is asked to "open" a message, it demonstrates that L_i was correctly constructed from L_{i-1}. The mix provides $seed_i$ and the correct corresponding image L_{i-1}. Anyone can then check that the value of $seed_i$ is correct with respect to Eq. 3.8:

$$\mathcal{D}_t^i = \{seed_i, D_t^{i-1}\}_{K_i} \tag{3.8}$$

and that the output L_i was appropriately computed from L_{i-1}, according to Eq. 3.7.

As trustees do not know beforehand which votes will be opened, trustees cheating to change the vote count will escape with a probability that decreases exponentially with the number of votes cheated on. If any four consecutive mixes do not collude, privacy is preserved in the absence of voter collusion. After all audits are complete, the election is certified.

3.6.1.2 Advantages and Disadvantages of the Ballot Based on Visual Cryptography

The ballot based on visual cryptography was one of the first developed in the context of an end-to-end voting system. Being the first, it possesses shortcomings with respect to practicality, but, at the same time, possesses excellent theoretical properties. In this section we describes its strengths and limitations.

Its strengths are as follows:

1. Because the ballot is essentially a pure black and white image containing the choices that the voter made, the content of the ballot can be anything: from names of the candidates to write-in votes, party symbols, rough sketches of the figures elected, or essentially any drawing. The font used to print the text on the ballot or the images that are printed on the ballot must possess sufficient redundancy so that they can be unambiguously interpreted when only every other pixel in the image is available. This is because ballots constructed by decrypting receipts contain only half of the pixels in the image. The other half of the pixels in the receipt are pseudo-random pixels that contain no information about the image and are discarded after checking the receipt for correctness.

2. The ballot based on visual cryptography can be used by illiterate voters via some graphical symbols that the voting machine presents to the voters. The availability of write-in candidates and the fact that they should be treated and counted no different then pre-set candidates can also be mandated by the rules of a particular jurisdiction, in which case the ballot based on visual cryptography would perfectly comply. The order of the candidates on the ballot can be fixed or can vary among ballots. Some jurisdictions impose a strict ordering, while others use different ballot styles in the same election, with the order of the candidates different on each style. Both situations can be accommodated by the ballot based on visual cryptography without interfering with any other part of the protocol.

3. The receipt creation is automatic; the voter does not have to take extra steps, or attempt to manually create a receipt by herself. Because the ballot is composed of two symmetrical parts, and either can be a receipt, the voter gets a receipt at the end of the protocol, without any special effort.

4. Because there are no physical cleartext ballots, either before the voter comes into the booth, or afterwards, no chain of custody is needed to protect the privacy of the ballots[2]. However, it is necessary to protect the private keys of the voting machine, since possession of the private keys implies knowledge of the pseudo-random bits used in ballot construction (see Eq. 3.2). It is recommended that the private keys of the booth be destroyed when the voting period ends (the keys could be generated when the period starts).

5. The dispute resolution process is easy: if a voter possesses a receipt that is incorrectly formed (but correctly signed) or the entire receipt does not appear correctly on the bulletin board, the voter holds irrefutable proof that something went wrong in the election. The voter only needs to present the original receipt as evidence of a problem.

This approach also possesses weaknesses, which are as follows:

1. The voting machine used by the voters to cast ballots knows the cleartext selections that the voters made. On the one hand this is an advantage, since the machine can compute an independent tally and report it when asked (e.g. at the end of the voting period or even during the voting period if partial tallies are desired). On the other hand, more importantly, this may be a privacy issue, since the machine has information that can be used to link the voters to the votes; for example, a time stamp on every vote can be associated with a time log of when voters arrive at the polling place; or, if voting machines have fingerprint readers incorporated for authentication purposes (as in Brazil) the fingerprints can be associated with the votes.

2. Voters are not familiar with the two layer voting interface. We are not aware of any usability studies performed on this interface.

3. Voter verification is a difficult process. First, the voter has to check every line on the ballot to make sure that the names of the candidates are correct, that the correct font, size and style have been used for printing. Second, the voter has to check that her physical receipt has been correctly posted on the SPBB; this implies that the voter should check that every other pixel on her receipt is identical to the posted image; additionally, she needs to make sure that a rather long string of ASCII text on her receipt is identical to that posted. While this

[2]None of the systems assume chain of custody for integrity

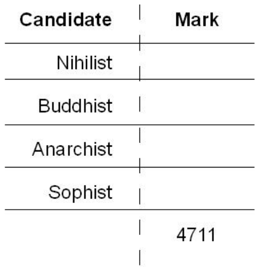
task can be made easier by requiring the voter to check random positions in both the image and the string, we think that not many voters will comply.

4. Implementing the ballot based on visual cryptography turned out to be very difficult in practice, for all practical US elections. On one hand, the size of the ballot (the number of contests and the number of candidates that can be selected in each contest) and the practical limitations on the size of the physical support on which the ballot (and the receipt) is printed, require that the print resolution be rather high. Having small pixels allows for more content on a limited printing surface. On the other hand, however, high resolution printing poses its own problems. Additionally, it makes alignment of the two ballot layers more difficult.

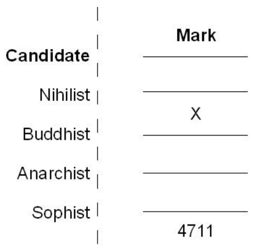
5. A manual recount is not possible as there is no permanent record of the clear-text choices that the voters make. Such a record could be produced by the voting machine, but this would lead to two records that the voter would have to compare, reducing the usability even further. It would create confusion regarding which of the two records was the legal one. In the case of a dispute, the election authority cannot resort to a manual recount, but would have to take some other remedying action (cancel and rerun the election, eliminate the incorrect records, inform the public and invite it to check the rest, etc.)

6. The management of a voting system that uses a ballot based on visual cryptography is difficult. Each voting machine needs its pairs of keys; neither a public key infrastructure nor an ad-hoc key authentication solution (e.g. the machine generates the pairs of keys at start-up and prints the public key on a piece of paper that the poll workers sign) is easy to manage. The setup of each machine for particular precincts (including ballot layouts and multiple languages), associated with the need for an independent trusted digital signature checker that can verify the authenticity of the receipts, only add to the complex management problem. Printers are also notorious for jamming, running out of ink or out of paper, thus the training of the poll workers would have to be extensive.

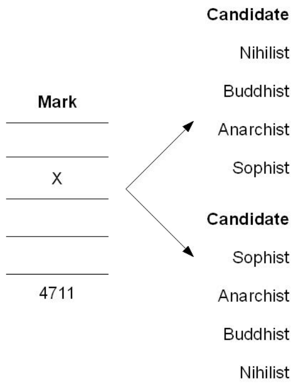
7. In terms of cost, the ballot based on visual cryptography is an expensive solution, for two main reasons: first, one machine per voting booth is needed (as opposed to one machine per precinct for precinct optical scan or even one machine per jurisdiction for central scan) and second, the machine itself is complicated enough that it would come at a high cost. A touch screen DRE machine already comes at a fairly high price (thousands of dollars), and a ballot based on



(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.

Figure 3.3: Prêt à Voter ballot

visual cryptography would need in addition a high performance printer, a state of the art shredder, and a hardware security module. The last piece of hardware itself is in many cases prohibitively expensive. costs thousands of dollars.

3.6.2 Prêt à Voter

Prêt à Voter [CRS05] is based on a simple and clever observation: if candidates do not appear in the same order on all ballots, the position of the mark next to a chosen candidate is itself an encryption of the vote. Moreover, the "encrypted" votes can be transformed into cleartext votes using a mixnet that is very similar to the one used by CVV in Sect. 3.6.1.

The ballot consists of a single piece of paper having a vertical perforation. To the left of the perforation is the list of candidates in a permuted order. The voter makes a mark next to the favorite candidate to the right of the perforation. See Fig. 3.3 for an example. After marking, the left side (containing the candidates) is separated and shredded and the right side is scanned and and made public. The voter gets to keep the right side as a receipt and checks it later to see if the public version of it is identical. If it is not, she has irrefutable proof of an error.

3.6.2.1 Prêt à Voter Details

The mixnet in Prêt à Voter consists of M mixes, each having a pair of keys $(K_i, \hat{K_i})$, where K is the public key corresponding to private key $\hat{K}$. To generate the ballots and print them in the first place, a trusted printing authority is needed. This authority has a pair of keys $(K_v, \hat{K_v})$.

Let $V = Alice, Bob, Carol, ...$ be the list of c candidates on the ballot.

Consider a single ballot. For each mix in the mixnet, the printing authority generates a (pseudo)random value $seed_i$ and uses it to create a permutation

$$\sigma_i = F(seed_i) \in \mathcal{P}_c \tag{3.9}$$

where $\mathcal{P}_c$ is the set of permutations of size c, and F is a pre-determined association rule between the random value and a permutation. All the permutations are then composed together

$$\sigma = \sigma_1 \circ \sigma_2 \circ \sigma_3 \circ ... \circ \sigma_{M-1} \circ \sigma_M \tag{3.10}$$

where $\circ$ represents the composition operator. The resulting permutation σ is applied to the list of candidates V, resulting in a permuted list of candidates V_σ, that is printed on the left side of the ballot.

The seeds may be generated pseudo-randomly as follows. If $K_V, \hat{K_V}$ is a pair of asymmetric keys of the entity generating the seeds, $seed_i = h(i, \{s\}_{\hat{K_V}})$, where h is a secure one way function and s is the serial number of the ballot. It can be proven that the seeds are constructed in a deterministic manner, and the seeds themselves do not constitute a covert channel.

The seeds are buried in a cryptographic doll:

$$\mathcal{D}_i = \{seed_i; \mathcal{D}_{i-1}\}_{K_i} \tag{3.11}$$

where $\mathcal{D}_0 = NULL$. The doll is signed to obtain $\{\mathcal{D}_M\}_{\hat{K_V}}$, and the signed doll is printed on the right side of the ballot. The doll is used by the mixnet to decrypt the receipt. Note that it is not necessary for the onion (the doll) to be printed on the ballot. The serial number of the ballot is sufficient to publicly identify the onion, which does not depend on the cast vote, but only on the order of the candidates on the ballot.

To ensure that candidate permutation on the ballot is consistent with the information provided to the mixnet, a printing audit is performed: each voter is allowed to choose two ballots, one to audit and spoil, and the other one to mark and cast. For the audited ballots, either the printing authority or the mixnet makes public all $seed_i \forall i\ 0 \leq i \leq M$, and it can be checked that Eq. 3.9, Eq. 3.10 and Eq. 3.11 are true and that the order of the candidates on the left side of the ballot is obtained by applying σ to V. The probability that the printing authority is cheating and does not get caught drops exponentially with the number of ballots cheated on.

After all the ballots are cast, the receipts are made public. After voters check that receipts on the SPBB are valid, the mixnet decrypts the votes. The input to the mixnet consists of pairs of the form $(w_M, \mathcal{D}_M)$, where w_M are the encrypted values (positions) of the chosen candidates. There are as many pairs as cast votes, but for clarity of notation we drop the index that would indicate the position of the ballot in the input and output of each mix. The reader should keep in mind that the mix is permuting the transformed inputs before outputting them.

The input for mix i is $(w_i, \mathcal{D}_i)$ and the output is $(w_{i-1}, \mathcal{D}_{i-1})$. The output $\mathcal{D}_{i-1}$ is constructed by decrypting $\mathcal{D}_i$ and removing $seed_i$

$$\mathcal{D}_{i-1} = \{\mathcal{D}_i\}_{\hat{K_i}} \setminus seed_i \tag{3.12}$$

where $\setminus seed_i$ represents removing the string $seed_i$ from the beginning of the first string. w_{i-i} is obtained by computing the inverse of the permutation obtained from $seed_i$ and composing it with c_i

$$\sigma_i = F(seed_i)\ and\ w_{i-1} = w_i\ o\ {\sigma_i}^{-1} \tag{3.13}$$

Note that, because permutation composition is not commutative, the order in which the permutations are composed is important.

As in CVV, the mixes are finally audited. The use of randomized partial audits results in a fast audit which is, however, not zero-knowledge. Each mix is required to demonstrate the correct processing of half of the inputs (or outputs), this is done by demonstrating the input-output connection and revealing the corresponding value of $seed_i$. The auditor can check the correctness of Eq. 3.13 directly and Eq. 3.12 by checking that

$$\mathcal{D}_i = \{seed_i; \mathcal{D}_i\}_{K_i} \tag{3.14}$$

3.6.2.2 Advantages and Disadvantages of the Prêt à Voter Ballot

The Prêt à Voter ballot addresses the major practical short-comings of the ballot based on visual cryptography. In particular, it addresses the high resolution printing and alignment problem. Additionally, it simplifies considerably the problem itself and the ballot presentation.

Its advantages are as follows:

1. The most common practical types of contests are supported by the Prêt à Voter ballot without any modification, including plurality voting, rank voting and approval voting.

2. Receipt creation is automatic; the part of the paper where the voter makes the marks becomes the receipt after it has been detached from the list of candidates and scanned. Checking the correctness of the receipt is easy: the voter simply checks if the positions marked are the same on the receipt and on the bulletin board. Dispute resolution is easy, a voter with a receipt that does not appear exactly on the bulletin board has sufficient evidence of an error.

3. In practice, the implementation is easy: the ballots can be printed on regular paper with any kind of printer (from ink jet printers to industrial presses), the scanner needed at the polling place can be of a low resolution, or even a mark sense scanner. The printing of ballots can be done off line, long before the election, such that printer jams do not interfere with voting procedures. The cost of the system is low, because a single scanner is sufficient per precinct, no matter how many voting booths the precinct has. If immediate feedback from the machine to the voter to warn the voter about potential over votes or under votes are not required, then a central scan may be sufficient. A high security scanner is essential, but the security of the printer is not as serious an issue.

4. The scanner that reads in the ballots at the precinct sees only the receipt that the voter takes home, which is also a matter of public record. The scanner hence cannot associate serial numbers, or voters, with cleartext votes. For the same reason, the scanner cannot compute an independent tally and report it at the end of the day, nor can a manual recount be initiated in case of disputes or fraud suspicion.

Its limitations are as follows:

1. A strict chain of custody is required to ensure that no one is able to link blank ballots with receipts, as such a link would reveal a correspondence between serial numbers and votes. After the voter receives and casts the ballot, however, a strict chain of custody is no longer required. Any particular marks that the voter makes on the receipt are irrelevant (even fingerprints or DNA), even if recorded by a coercer, because the receipt, without information on the corresponding encryption function (candidate order) does not reveal information on the vote.

2. The random order of the candidates is an essential ingredient of the Prêt à Voter ballot. This requirement may come into conflict with regulations that specifically mandate a certain order of the candidates on the ballot (such as alphabetical), or that simply require that all the ballots have to look the same to the human eye. Some jurisdictions impose multiple ballot styles, where the order of the candidates is different for each style to compensate for the fact that it has been reported that the candidate that appears at the top of each race tends to get more votes solely based on position.

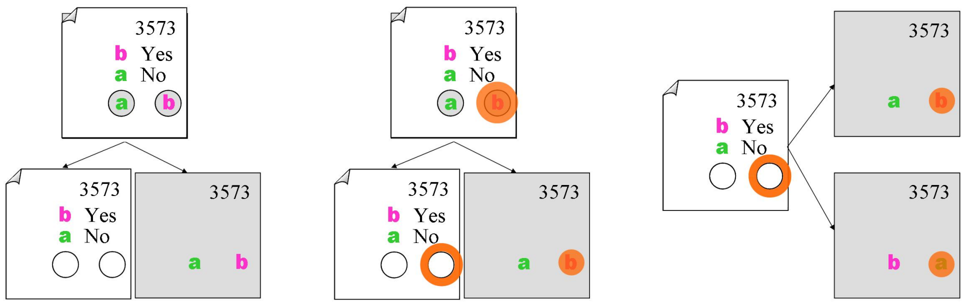
3. While the interface is inherently familiar to the voters (make a mark next to the favorite candidate), it is often the case that political parties distribute sample voted ballots (via mail or in front of the polling places) to help sympathizers of that party to fill in their ballots; some voters may tend to associate geometrical marked positions with votes for a candidate (e.g. fill in the oval for the top candidate) and the randomization of candidates will interfere with this association, creating an unwanted butterfly ballot.

4. While write-in candidates can technically be treated with this system, the application to write-in candidates is cumbersome and is not straightforward.

3.6.3 PunchScan

PunchScan was first presented by David Chaum in September 2005 [Cha05b]. The first detailed description is due to the author and Ben Hosp [PH06]. This section presents the basic version of PunchScan, as described in [PH06]. Sect. 5.2 presents enhancements to the basic system to improve on flexibility, integrity assurance and other important aspects.

A PunchScan ballot consists of two stacked sheets of paper. The top page of the ballot has holes in it, and the information on the bottom page can be read through the holes. The top page contains contests (i.e.: ballot questions or offices) and the



(a) A sample PunchScan ballot. When the two pages are overlayed, 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.

Figure 3.4: PunchScan ballot

corresponding options or the candidates' names. Each option has a symbol assigned to it, and the assignment of symbols to answers varies from ballot to ballot. On the bottom page of the ballot, there is a list of the symbols, in an order that also differs from ballot to ballot, in a manner that is independent of the order on the top page. The top and the bottom ballot pages are aligned in such a way that when they are overlaid, for every question on the ballot, the symbols from the bottom page are visible through the holes made on the top page (see Fig. 3.4(a)).

The voter uses a dauber (a pen that leaves a disk of ink on the paper when it makes contact) to mark the selection of candidates. The diameter of the ink disc is greater then the diameter of the hole punched through the top page, thus the dauber leaves a mark on both the top and bottom ballot pages. Fig. 3.4(b) contains a ballot voted for "Yes". If one assumes that (a) the order of the symbols on each of the two pages of a ballot is uniformly random and (b) the orders of symbols on the top and bottom pages are independent, the accuracy of a guess regarding the chosen candidate does not improve on viewing only one page. Fig. 3.4(c) has the right answer selected on the top layer; depending on which possible bottom layer is this ballot's actual bottom layer, that mark could represent a vote for "Yes" or a vote for "No", each with a probability of 0.5, if nothing else is known about voter choices.

3.6.3.1 Chronological Description of the Election Process

There are three phases of the election process:

- the preelection phase (labeled B for *Before*)
- the election phase (labeled E for *Election*)
- the postelection phase (labeled A for *After*)

The preelection phase The preelection phase is preparatory and allows for the setup of the election and of the integrity proofs. During the preelection phase, the ballots are generated, printed and audited. Also, the information that allows for the recovery of the choice from one page of the ballot is generated and checked. The chronological order is as follows:

B.1 The election authority generates ballots and commits to them.

B.2 The election authority generates and commits to the information necessary for decrypting one page of the ballot when the other one is destroyed.

B.3 The candidates challenge the election authority and ask to see some of the ballots (say half), along with the information from B.2.

B.4 The election authority provides the requested ballots, and opens the commitments associated with them, thus spoiling them.

B.5 The candidates check to ensure that the commitments are consistent with the opened ballots.

Election day On election day, the voters go to their assigned polling places, authenticate themselves as legitimate voters, and get a ballot from the election officials.

E.1 The voter is given a sealed ballot.

E.2 Without seeing the order of the symbols on either page, the voter commits to the page that will be kept (e.g by making a special mark on the other page).

E.3 The voter uses the dauber to mark the hole that has the symbol associated with her favorite candidates on the ballot.

E.4 The voter separates the two pages, keeps the one chosen in [E.2] and destroys the other.

E.5 The surviving page is scanned, and the positions of the marks are recorded and made public. Henceforth, all references to “ballot” will refer to this surviving page.

In an earlier version, the voter chose which page to keep after seeing and marking the ballot. This allowed for the possibility of a coercion attack described by John Kelsey [KRMC07]; this attack is not possible with the modified procedure described above.

The postelection phase After all the polls close, the election is audited and proofs carried out to ensure the integrity of the election. The chronological order of the events following an election is as follows:

A.1 Any voter can go to the election authority web site, enter a serial number for her ballot, check that the ballot is there, and that it accurately resembles the page she possesses.

A.2 The election authority processes all ballots to produce decrypted versions, along with a partially decrypted form of all the ballots.

A.3 The candidates ask to see some of the transformations from the original ballots to the partially-decrypted forms, and some of the transformations from the partially-decrypted form to the clear form.

A.4 The election authority replies to the challenges made by the candidates in [A.3].

A.5 The candidates check to see if the reply of the election authority is consistent with the commitments made in the preelection phase [B.2] and with the information made public in [A.2].

3.6.3.2 Description by roles

In an election that uses PunchScan, like in any election, there are three main actors: the voters, the election authority and the auditors (or observers). In a typical election today, however, the election authority and the auditors are one and the same entity, because only internal audits are performed, and only those with privileges are able to check the correctness of the audits. In an election using PunchScan, there is a clear delimitation between the auditors and the ones running the elections and handling ballots. Starting from the observation that the goal of an election is to convince the losers that they indeed lost, we suggest that the candidates themselves would make good auditors, as it is in their best interest (especially in the interest of the candidates that lost) to check that the election was conducted properly. We now present each role separately and describe its responsibilities.

The voter On Election Day, a voter comes to the assigned polling place and authenticates herself as a legitimate voter. She gets a dauber and a ballot, and before seeing it, commits to the page that she will keep. She enters a private voting booth. She chooses her favorite candidates by making a mark with the dauber on the hole that has the symbol associated with her favorite candidate. She then keeps the page she committed to keeping, and shreds the other one. Then, she scans the kept page. She may walk out of the polling place with this page, which serves as her (encrypted) receipt. Later, she can go to a web site, type in the serial number of her ballot, and check that the ballot is there. No other checks are required from the voter; further, she can delegate this check to someone she trusts.

The election authority In the preelection phase, the election authority decides the format of a canonical ballot. This is the one from which all the other ballot variants will be generated. Also, the canonical ballot is used to recover the choices of the voters, after one page of the ballot has been destroyed.

The election authority generates at least twice the number of ballots needed in the election, and commits to them (making the commitment public; the ballots themselves remain secret). It also generates and commits to information necessary to recover the intent of a voter from one page of the ballot.

In response to the preelection challenge [B.3], the election authority discloses all the information about half (or a significant fraction) of the ballots (thus spoiling them). This allows the candidates to check the commitments and ensures (with high probability) that all the ballots have been generated consistent with the information that is required to decrypt them from partial information.

After the election, the election authority posts partially decrypted ballots (shuffled) and cleartext ballots (further shuffled). To prove that both decryptions (partial and final) were performed correctly, for each vote the election authority will reveal either how it transformed the voted ballot into a partially decrypted one, or how it transformed a partial decrypted ballot into a cleartext one, but not both for the same ballot. The auditors choose which part will be revealed, and the chances of a cheating election authority not being detected decrease exponentially with the number of votes cheated on.

The candidates The candidates may play the role of auditors, challenging the election authority during the preelection and postelection phases, and checking that the replies are consistent with the commitments.

3.6.3.3 A sample election using PunchScan

We describe a simple example: the election consists of a single binary contest; the voters vote "Yes" or "No". The election authority decides that, in the canonical top page, the symbol "a" is associated with "Yes" and the symbol "b" with "No" on the top page. In the other type of top page, sometimes referred to as the "flipped" top page, the symbol "a" is associated with "No" and the symbol "b" with "Yes". The election authority also decides that the order is "a" "b" on the top page. There are two types of bottom pages, again the canonical one, with "a" followed "b", and the flipped one. The canonical ballot is presented in Fig. 3.5. Fig. 3.6(a) contains all the possible top and bottom pages. Any top page can be combined with any bottom page to produce a ballot as seen in Fig. 3.6(b). The four types of ballots are equally likely.

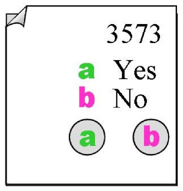
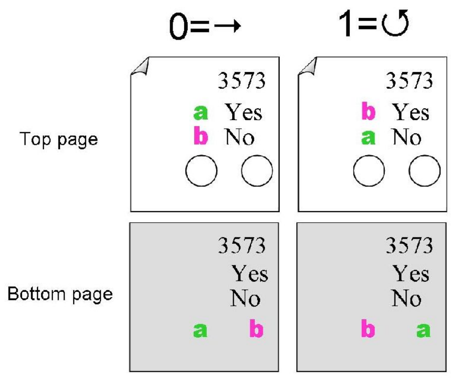
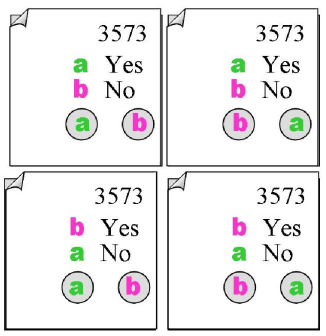


Figure 3.5: The canonical ballot for a Yes/No contest



(a) All possible pages of a Yes/No ballot



(b) All possible ballots containing Yes/No contest

Figure 3.6: PunchScan's ballot

$\oplus$	Canonical	Flipped
Canonical	Canonical	Flipped
Flipped	Flipped	Canonical

Table 3.1: PunchScan simple example of Flipped / Canonical ballot logic

A canonical ballot is one in which a mark on the left hand side is a vote for “Yes”. A flipped ballot is one in which a mark on the left hand side is a vote for “No”. Of the four ballot types in the figure, two are canonical ballots, and the other two are flipped ballots. A canonical top page combined with a flipped bottom page results in a flipped ballot. All the possibilities are in Table 3.1. The decryption of the vote (a mark on the left hand side, say) depends only on whether the entire ballot is flipped or canonical, a fact that cannot be determined by looking at one page. If canonical ballots, top and bottom pages are represented by 0, and the others by 1, the parity of the ballot is the *exclusive or* of the parities of the two pages.

Denote “Yes” by 0, and “No” by 1. Let a mark on the left hand side be denoted 0, and one on the right hand side 1. This is the encrypted vote. To decrypt the encrypted vote, it is necessary to know if it came from a flipped or canonical ballot, to know if it should be flipped or not. In PunchScan, this information is further split into two flip/non-flip operations (flip1 and flip2) for each ballot. When combined, these operations transform the ballot page to the canonical ballot. The information is split so that one half can be made public for auditing purposes. The relation that has to hold between the pages of the ballot and the information used for recovering is: top $\oplus$ bottom = flip1 $\oplus$ flip2.

The election authority makes public commitments to the ballots and to flip1 and flip2. The candidates choose half of the ballots at random, and the election authority makes public the requested ballots along with the flip1 and flip2 for each ballot. Anyone can check that the equation top $\oplus$ bottom = flip1 $\oplus$ flip2 holds, and that “top” and “bottom” correspond to the values of the printed ballots. Only the ballots that were not made public in this phase (pre election) will be further used in the election.

During the election phase, the election authority publishes all the marked pages (half ballots) as voted on by voters. After the election, it publishes the intermediary state of the (shuffled) ballots (ballot $\oplus$ flip1) and the decrypted (further shuffled) ballots (ballot $\oplus$ flip1 $\oplus$ flip2). These are commitments to the values of flip1 and flip2 used in the decryption of the voted half ballots.

During the postelection phase, the election authority is asked to open either flip1 or flip2 but not both, since opening both would allow the linking of a voted ballot to the corresponding decrypted one. Also, it is necessary that the partially-decrypted ballots and the decrypted ones be in a random order (distinct from each other and from the order of the voted ballots).

The election authority defines the following tables:

- P (for **Print**)
- D (for **Decrypt**)
- R (for **Results**)

The P table is indexed by ballot serial number and contains the top page (P_1), bottom page (P_2), and space for the filled-in vote (P_3, to be entered after the election). It also contains commitments to P_1 and P_2. P_3 and the commitments are made public. Some other values are audited as described later.

The D table contains the first (D_2) and second (D_4) flip, the partially-decrypted vote (D_3) to be filled in during decryption (the encrypted vote flipped according to D_2), and information to connect it with the P table (D_1) and the R table (D_5). It also contains a commitment for each row of D, as well as a commitment for columns D_1 and D_2 (together), and another commitment for columns D_4 and D_5 (together). Only D_3 and the commitments are made public. Some other values are audited as described later.

The R table contains the cleartext votes (after postelection decryption).

For example, consider an election with six votes. The clear data in all the tables is in Table 3.2. (No single person will ever see all of this information.) Before the election, but after the election authority has made the commitments, the tables look as in Table 3.3.

The candidates challenge the election authority to open at random half of the ballots, say the ones numbered 2, 4 and 5. The election authority reveals the requested information, and the tables look as in Table 3.4. Ballots 2, 4, and 5 now cannot be used in the election and are excluded from any further representation of the tables (see Table 3.5).

Assume that the voters mark their ballots as follows: on ballot 1, the left hole is marked, and the top page is chosen; on ballot 3, the right hole and the bottom page are chosen; on ballot 6, the left hole and the top page are chosen. Because the

Ballot ID	P_1	P_2	P_3	CP_1	CP_2
1	ab	ab		$C_{1,1}$	$C_{1,2}$
2	ab	ba		$C_{2,1}$	$C_{2,2}$
3	ba	ab		$C_{3,1}$	$C_{3,2}$
4	ba	ba		$C_{4,1}$	$C_{4,2}$
5	ab	ba		$C_{5,1}$	$C_{5,2}$
6	ba	ab		$C_{6,1}$	$C_{6,2}$

D_1	D_2	D_3	D_4	D_5	DC
6	$\rightarrow$		$\circlearrowleft$	5	C_A
5	$\circlearrowleft$		$\rightarrow$	4	C_B
2	$\circlearrowleft$		$\rightarrow$	1	C_C
1	$\circlearrowleft$		$\circlearrowleft$	3	C_D
4	$\rightarrow$		$\rightarrow$	2	C_E
3	$\rightarrow$		$\circlearrowleft$	6	C_F
$CD_{1,2}$			$CD_{4,5}$		

R_{id}	R_1
1	
2	
3	
4	
5	
6	

Table 3.2: PDR tables as the election authority sees them, with all the information available. The tables are properly formed, because, for all the ballots, $D_2 \oplus D_4$ correctly represents whether P_2 is a flipped version of P_1 or not. For example, for ballot number 3, on the top page, "a" is associated with "Yes", and b with "No". On the bottom page, the order is "ba", thus P_2 is a flipped version of P_1. In the D table, in the row corresponding to 3, we have $\rightarrow \oplus \circlearrowleft =$ flip. For ballot 1, $C_{1,1}$ is a commitment to P_1, $C_{1,2}$ is a commitment to P_2 and so on.

Ballot ID	P_1	P_2	P_3	CP_1	CP_2
1				$C_{1,1}$	$C_{1,2}$
2				$C_{2,1}$	$C_{2,2}$
3				$C_{3,1}$	$C_{3,2}$
4				$C_{4,1}$	$C_{4,2}$
5				$C_{5,1}$	$C_{5,2}$
6				$C_{6,1}$	$C_{6,2}$

D_1	D_2	D_3	D_4	D_5	DC
					C_A
					C_B
					C_C
					C_D
					C_E
					C_F
$CD_{1,2}$			$CD_{4,5}$		

Table 3.3: PunchScan PD tables in the preelection phase, as the public sees them.

canonical ballot is "ab","ab" (that is, "ab" on both pages), left is associated with "a", and right with "b". The voters' choices eventually end up in P_3, and when they do, each row describes what can be learned through knowledge of the ballot page chosen by the voter.

The election authority performs the first flip to ballots 1, 3 and 6 to obtain the partially decrypted ballots in D_3, and the totally decrypted ballots in R_1 (see Table 3.6). The ballots in both D and R are shuffled independently, so it is not possible to link rows among tables P, R and D. During the postelection phase, the auditor asks the election authority to open either the left or the right side of D (but not both). If the election authority cheats, the auditor will catch it with probability 0.5 (for a higher probability, multiple independent instances of the D table can be used). In our example, suppose the auditor chooses the right side. The election authority then reveals D_4 and D_5. The auditor can now check that $D_3 \oplus D_4 = R_1$, and that the commitment $CD_{4,5}$ to the columns D_4 and D_5 is valid.

In this simple example, there is only one question per ballot. The situation be-

Ballot ID	P_1	P_2	P_3	CP_1	CP_2
1				$C_{1,1}$	$C_{1,2}$
2	ab	ba		$C_{2,1}$	$C_{2,2}$
3				$C_{3,1}$	$C_{3,2}$
4	ba	ba		$C_{4,1}$	$C_{4,2}$
5	ab	ba		$C_{5,1}$	$C_{5,2}$
6				$C_{6,1}$	$C_{6,2}$

D_1	D_2	D_3	D_4	D_5	DC
					C_A
5	$\circlearrowleft$		$\rightarrow$	4	C_B
2	$\circlearrowleft$		$\rightarrow$	1	C_C
					C_D
4	$\rightarrow$		$\rightarrow$	2	C_E
					C_F
$CD_{1,2}$			$CD_{4,5}$		

Table 3.4: PunchScan PD tables after the election authority has replied to the request to open ballots 2, 4, and 5.

Ballot ID	P_1	P_2	P_3
1			
3			
6			

D_1	D_2	D_3	D_4	D_5
$CD_{1,2}$			$CD_{4,5}$	

Table 3.5: Ballots that can be used by voters on election day. The other ballots were spoiled during the preelection phase. The row commitments are not shown anymore because they won't be checked, since no other complete row will ever be opened.

Ballot ID	P_1	P_2	P_3
1	ab		a
3		ab	b
6	ba		a

D_1	D_2	D_3	D_4	D_5
		a		
		b		
		b		
$CD_{1,2}$			$CD_{4,5}$	

R_{id}	R_1
3	a
5	b
6	a

Table 3.6: PunchScan PDR tables after the polls close. One cannot say what row in the D table corresponds to what row in the P or R table, because the rows are shuffled. Thus, the secret ballot principle is satisfied.

Ballot ID	P_1	P_2	P_3
1	ab		a
3		ab	b
6	ba		a

D_1	D_2	D_3	D_4	D_5
		a	$\circlearrowleft$	5
		b	$\circlearrowleft$	3
		b	$\circlearrowleft$	6
$CD_{1,2}$			$CD_{4,5}$	

R_{id}	R_1
3	a
5	b
6	a

Table 3.7: PunchScan PDR tables after the postelection audit. The election authority was asked to open the right side of the D table. Anyone can check that the partially decrypted result transformed by D_4 gives the result in R ($D_3 \oplus D_4 = R$), so the election authority did not cheat. Also $CD_{4,5}$, the commitment to D_4 and D_5, is checked. Note that there is still no link between P and R, so privacy is maintained.

comes slightly more complicated if this is not the case. PunchScan works well for multiple-question ballots, but the decrypted ballots preserve the “cross-question” relationships: for example, if 90% of the people who voted for Alice for Governor also voted for Bob for President, the results will reflect this. This can be addressed by having different ballots (different serial numbers) for different races.

3.6.3.4 Advantages and Disadvantages of the PunchScan ballot

The PunchScan ballot has similarities with both the CVV and Prêt à Voter ballots. On the one hand, as with CVV, it has two parts, and only when the parts are overlaid is the vote visible. Further, like CVV, the voter can choose any of the parts as a receipt. On the other hand, the alignment problem is easier than in CVV.

Its advantages are as follows:

1. The PunchScan ballot supports most practical contests.
2. Receipt creation is automatic: the voter simply separates the two pages and keeps one. Voter verification and dispute resolution are both easy, since the only thing the voter has to do is to check that the correct hole appears marked on the website and the order of the symbols on her receipt is the same as presented on the web site.
3. Voting system and election administration is relatively easy, and the cost is low: regardless of the number of voting booths, a precinct needs one scanner, one computer and one shredder.
4. PunchScan can produce ballots keeping the order of the candidates fixed on all ballots. This can be a very important requirement if the rules of the voting jurisdiction dictate that the list of candidates should be ordered in a pre-determined manner.

Its disadvantages are as follows:

1. Like the Prêt à Voter ballot, the PunchScan ballot format does not easily support write-ins.
2. There is a need for a strict chain of custody before the ballots reach the hands of the voters to protect the privacy of the voters.
3. A manual recount is not possible since there is no cleartext ballot that gets kept at the polling place, or anywhere else.

4. Producing the ballots is more expensive than regular optical scan ballots or Prêt à Voter ballots, since two pages are needed per ballot, the top pages need to be drilled with extremely high precision and both pages have to printed on printers that allow pre-punched paper and respect the exact coordinates given by the computer. The association between top and bottom pages needs to be maintained till the ballot is used; however, this can be done, for example, by printing the entire ballot on a single sheet of paper, double sided, and then folding it, or by using a fancier printer with two paper trails and a stapling function.

5. The familiarity of the user interface is questionable [BJK$^+$]: the indirection step may not be correctly understood by all voters, and it is possible that voters will mark ballots differently then they intended. We are not aware of any usability studies of the PunchScan ballot.

Chapter 4

A Framework for Voter-Verifiable Voting Systems Based on Mixnets

All the systems described in Sect. 3.6 are based on mixnets; either classical onion mixnets or specialized mixnets, as in PunchScan. In this chapter we describe how each of the systems may be viewed as a specific instance of a more general system. In particular, we break down the monolithic voting systems and the associated processes into a sequence of components and steps respectively, and describe the voting, tallying and verification processes at an abstract level that represents all three systems. In later chapters we provide more mathematical detail on the general system, and demonstrate how individual examples are special instances of it. This work is the first to treat this class of systems at a general level. A large part of the work has been published in the Cryptologia journal [PV] and at WOTE 2008 [PV08a].

We observe that the voter is involved only in a small number of steps, and that the ballots are handled independently of the manner in which receipts are counted. There are two major roles played by system components: one component produces the ballots and interacts with the voters, while the other tallies the votes in a publicly-verifiable manner. We call the first component the front-end, and the second the back-end; in the examples in Sect. 3.6 the back-end is the mixnet. Intuitively, the front-end transforms a cleartext vote input by the voter to an encrypted vote, and the back-end transforms the encrypted vote back into a cleartext vote. Both components function in a verifiable manner: to be voter verifiable, the front-end performs the transformation in a manner that a human voter can check, the back-end performs a universally-verifiable decryption. One can construct both a stand-alone back-end and a stand-alone front-end, each receiving input in a standardized format, and providing output in a standardized format as well. One may then construct new voting systems by constructing front and back-ends that have not been combined previously. This dissertation thus applies the principles of decoupling, which has proven to be successful in other aspects of computer science (the ISO/OSI network stack is build on decoupling principles, for example X.200 [Uni94]), to voting systems.

The front-end of a tallied as cast voting system (see Definition 13) is the part of the system that interacts with the voters and ensures the voters that their votes were recorded as cast. The back-end is the part that ensures that all the votes were counted as recorded, and that can be verified by anyone (not necessarily voters). The front-end is voter-verifiable while the back-end is universally verifiable.

Our framework requires the existence of a secure public bulletin board (SPBB, see 10). We now describe the detailed steps that are followed by all the voting systems in our framework. In some specific cases, some of the steps described bellow are skipped, while other steps may involve some detailed description. We assume proper voter registration and authentication procedures, and that each voter gets the ballot style that she is allowed to (and entitled to) vote on; these aspects, while very important, are out of the scope of this dissertation.

In general, a voting system conforming to our framework works as follows:

1. The back-end is set-up.

2. The back-end gives any necessary information to the front-end.

3. The front-end is set-up.

4. Both the back-end and the front-end are checked for correctness.

5. The front-end prints the ballots and distributes them to the polling places.

6. Ballots are assembled.

7. Ballots are handed out to the voters and voted upon. The front-end proves that the ballots were printed correctly.

8. Each voted ballot is disassembled and a receipt is produced for the voter; this receipt is an encryption of the vote. The encryption is correct if and only if the ballot was printed correctly.

9. The front-end sends the receipt to the back-end and the back-end publishes the receipts.

10. The receipts possessed by the voters are checked against the published receipts.

11. The back-end processes (decrypts) the receipts and publishes cleartext ballots.

12. The back-end proves that the published receipts decrypt to the cleartext ballots.

We do not make any assumptions about the nature of the back-end and front-end authorities. They may be centralized or distributed authorities, distinct or not. The authorities can be recreated at specific moments in time when they are needed, or they can exist temporarily.

In step 1, the back-end may secretly generate keys, generate some election data and publish commitments to it. Every back-end must specify how it implements this step.

In step 2 we assume there exists a private channel used by the back-end to send all the necessary data to the front-end. This data may be public, as in the public keys of the back-end, or it can be confidential, as having this data public may later result in the compromise of the privacy for some votes. It may not be necessary for the back-end to give to the front-end all the data that was generated in step 1.

In step 3 the front-end may secretly generate some keys or some data, possibly based on the confidential information that it received in step 2, and it may publish some commitments to it. Every front-end must specify how it implements this step.

In a scenario where the front-end has to generate data in step 3 based on the data it got from the back-end in step 2, there is the possibility that the front-end may completely ignore the data it got from the back-end and decide to simply generate some random data. This may have severe consequences on the integrity of the system. To ensure that something like this does not happen, step 4 is of paramount importance. This is the first audit of the system, that may take place well in advance of the election. It is important that this verification step is done in a public manner. The verifier should not have to be present at a certain place or time. Anyone that checks the records on the SPBB must be convinced that the proof is sufficient. It is typically sufficient that the proof guarantees a high level of accuracy, even though it may not guarantee 100% accuracy; in most practical cases a level that is close to 100% is sufficient. Both the front-end and the back-end may be involved in this step, as it is meant to ensure that the entire system is set up correctly. Every combination back-end/front-end must specify how it implements this step. This step is not needed where step 3 does not depend on step 2

If the front-end produces blank ballots that will be distributed to the polling places, this operation takes place in step 5. The nature of the communication between the front-end and the polling place may vary from system to system. Some systems may require that a private communication channel be used to protect the secrecy of the ballots, while for others, a public channel may suffice. Every front-end must specify the nature of the channel used.

If the ballots consist of multiple parts (e.g. layers), the pieces are assembled in step 6. This step may be performed by a poll-worker if only certain combinations of parts are valid (e.g. parts with the same serial number), or by the voters if the parts can be randomly assembled to form a valid ballot.

Step 5 can be viewed as the most important step of the protocol. It is where the voter expresses her choices. First, the voter may be handed a specific ballot, or the voter may have the option of choosing a random ballot from a pile of available ballots. The link between the voter and the ballot that she receives can be broken in this way; however, the privacy of the system should not rest on this assumption. Because the ballot is the means for vote encryption, it is important that the ballot obtained by the voter is made in accordance with the data generated by the front-end in step 3; as the ballots were printed after the check in step 4, the printed data may be corrupted. The printing correctness check ensures with high probability that all of the ballots were correctly printed. Each front-end specifies how the voter can check if the ballot that she got is correctly printed. Every front-end must also specify how the ballots should be marked.

In step 8 the information that will become the voter's receipt is separated from the rest of the ballot. The voter is allowed to keep some information from the ballot; this could be part of the ballot itself. This information is the encrypted ballot, and it becomes the receipt that the voter can keep. All the other information/parts of the ballot are securely destroyed by the voter. Each front-end should specify how the ballots are disassembled; it should also specify the form of the receipt.

Step 9 is a simple step performed by both the front-end and the back-end. The front-end sends all the receipts (or copies of the receipts) it collected to the back-end, and the back-end makes them public on the SPBB.

In step 10, anyone that has access to a valid receipt can check the records on the SPBB and see that all the information on the receipt is correctly posted. It is not necessary that the check be performed by the voters themselves. We discuss later the two types of receipt that are possible in our frameworks. We also discuss which entity can perform the check, this depends on the type of receipt provided by the front-end. The check must ensure with high probability (not necessarily certainty) that all the receipts have been correctly posted. When a voter sees her receipt correctly posted on the SPBB, she can be assured that: (a) the front-end correctly read and recorded her receipt, (b) the chain of custody from the front-end to the back-end was unbroken, and (c) her receipt forms part of the input to the back-end.

In step 11 the back-end processes all the receipts on the bulletin board, gets cleartext votes and publishes them so that anyone can tally them to produce the election results. The back-end may also produce intermediary data and make it publicly available. The processing of the receipts, which results in the production of the intermediary data and the cleartext ballots, is performed secretly by the back-end, using secret data. Only the results of the processing—the cleartext votes and the intermediary data—are made public on the SPBB. Each specific back-end must specify how the cleartext ballots and the intermediary data are produced.

As receipt processing is secret, the back-end could falsely claim that a set of cleartext ballots represents the input set of receipts. Step 12 provides a proof that, with high probability, the set of cleartext ballots is obtained by decrypting the set of public receipts. The intermediary data published by the back-end in step 11, and the data published by the back-end at set-up, in 1, may be used for this purpose. Anyone checking the records on the SPBB must be convinced of the equivalence; that is, verification that the decryption was correct should not require access to secret data. In order to preserve privacy, it is important that no receipt be linked to a cleartext vote during the verification.

It is important that, whenever possible, everyone who wishes to perform checks on the integrity of the tally may do so. In steps 4 and 12 the verifications consist of checking the public data on the secure bulletin board against itself, to see if it is consistent with previously published data. Thus anyone can perform these checks; as long as the audit challenges have been constructed in a public manner (see [CEA07] for a way to do this), these steps provide universal verification. The other two steps that involve verification, step 7 and step 10, can only be performed by the voter, or a trusted organization that the voter delegates authority to, as the verification involves an event that has occurred in the booth that only the voter bears witness to (the casting of the ballot and the generation of the corresponding receipt). Our framework assumes that a statistically significant fraction of the voters will perform these checks; we also assume that the lower the winning margin the more voters will check.

The above approach to checking the integrity of the election is fundamentally different in three ways from any voting system that is currently used in any public election. First, instead of checking the voting equipment, each election is checked individually. In such a setting, certifying election equipment is not necessary; certifying each election is. This is in accordance with the notion of software independence, discussed in Sect. 2.3. It does not mean that we can have equipment that simply fails, or mediocre bug-riddled software; even though these would be detected by the checks that our framework performs, it is best if the election runs smoothly.

Second, since all the data needed to check the election is available on the SPBB, anyone can play the role of auditors, and do not need privileges to be able to check an election by themselves. Individuals need not register as observers, be part of privileged organizations, pay fees, or have background security checks. The level of transparency is maximal in this sense.

Third, a single individual may provide proof that the tally is incorrect; for example one auditor notices that one record from the data published in step 12 is not consistent with the record published for the same data in step 1. On the other hand, in other voting systems, there needs to be an agreement that something went wrong; those in charge of ensuring election integrity may vote, and the majority may decide whether the election was rigged. In our framework, if only a single person calls into question the count, and that single person can pinpoint the inconsistency in the proof, then this is sufficient to prove that things did indeed go wrong. While the verifiability guarantees are probabilistic, each front-end and each back-end should specify the level of assurance that is guaranteed in each of the four checks that occur.

It is also important that the integrity checks do not have to be made at a single moment in time. The checks can occur at any time: the auditors can check the SPBB at any time and see that the data on it is consistent; a voter can go and check the SPBB as many times as she wishes.

We present a very general mathematical model for the functionality of each of the two components. The front-end enables the voter to encrypt her vote without access to a trusted computational device. The ballot is presented in a manner that enables this. Associated with each ballot, which is identified by serial number, is an encryption function, i.e. the front-end implements a mapping $\mathcal{F} : \mathbb{Z}_n \times V \rightarrow \mathcal{W}$, where V is the set of cleartext votes, $\mathbb{Z}_n$ the set of ballot serial numbers, and $\mathcal{W}$ the set of encrypted votes. For the voting systems to be voter-verifiable and for it to be possible for voters to encrypt their votes by simply filling up the ballot, the encryption function should be one that the voters can easily understand; for example, it may simply be a substitution cipher (a particular cleartext vote on a given ballot is substituted with another symbol/value). The encrypted vote is the receipt that the voters get to keep at the end of the voting process. In order that the receipt be correctly decrypted by the back-end, the correspondence between serial number and encryption function should be known to the back-end. Additionally, so that the front and back-end may not collude to rig the election, any information communicated between the two for the purposes of correct decryption should be committed to. Further, as the voter does not know what the back-end believes is the encryption function used to encrypt her vote (and the back-end will perform the decryption according to what it believes is the encryption function) the front-end must prove to the voter that the encryption function use in printing the ballots is, with high probability, those communicated to the back-end. This is typically done with what is known as a *printing audit*. Note that the receipt the voter takes home should bear no indication of how the voter voted; i.e. looking at only the receipt, it should be equally likely that the voter choose as her cleartext vote any of the elements in V. This is necessary for coercion resistance.

The back-end decrypts the set of votes in a verifiable manner; i.e., the back-end implements a mapping $\mathcal{D} : \mathcal{W} \rightarrow \mathbb{Z}_n \times V$ (which can be views as the inverse of $\mathcal{F}$, even though mathematically this is not correct). Notice that it should be impossible for an observer (and ideally for the back-end also) to link with certainty a cleartext vote (an output of the back-end) to a particular receipt (an input). Hence, typically, the cleartext vote will not contain the serial number of the ballot used to cast it. The back-end must be universally verifiable: anybody must be able to convince himself or herself that the back-end correctly transformed the set of input receipts into the set of output cleartext votes. None of the receipts should be omitted, injected or modified.

We present the chronological evolution of voting systems that inspired this work. It all started with CVV in 2004, a front-end based on visual cryptography combined with an onion mixnet back-end. It had great theoretical properties, but the front-end was difficult to manufacture and to use in practice and the back-end was slow. Prêt à Voter was introduced in 2005, keeping the onion mixnet back-end and introducing a new front-end that was more user friendly and very easy to manufacture. PunchScan followed, also in 2005, bringing both a new front-end that allowed for the order of the candidate to be the same on all ballots and a new back-end that had great performance. Initially, in 2007, Scantegrity brought a new front-end that was an overlay to any regular optical scan voting system; the back-end was a "punchscanian" mixnet. In 2008, Scantegrity brought a new back-end that solved the Italian attack for both intra and inter-contest correlations. In the same year, Scantegrity II improved the front-end of Scantegrity in terms of privacy and resolution dispute. Also in 2008, Essex [ECA08] proposed Aperio, a new back-end that is very easy to describe to non-technical people, and keeps the privacy set of any ballot at maximum.

We expect that in the future, the evolution of end-to-end voting systems to follow the same trajectory: improving the front-end of an existing system, followed by the improvement of the back-end, followed by the improvement of the front-end again, and so on.

Chapter 5

Back-ends

This chapter presents three back-ends, and formally describes privacy and integrity guarantees. We present a summary of the widely known onion based mixnets [Cha81], an in-depth analysis and generalizations of the punchscanian mixnets, and a simplification of the punchscanian mixnet, a new back-end based on pointers.

5.1 Onion mixnets

This section describes the manner in which an onion mixnet implements the steps of the back-end of the general voting systems described in Chapter 4. For a description of how onion mixnets work, and a simple mathematical model, the reader is invited to read Sect. 3.4.3. The mixnet literature spans twenty-five years, and covers formal mathematical models as well as methods for proving correctness.

5.1.1 Onion Mixnets as Back-Ends

We now describe how an onion mixnet implements the steps that it is responsible for in the protocol described in Chapter 4:

1 The back-end is set-up: each mix in the onion mixnet generates a pair of asymmetric keys, $\{K_i, \hat{K_i}\}$.

2 The back-end gives any necessary information to the front-end: each mix sends its public key to the front-end (via a public channel).

4 Both the back-end and the front-end are checked for correctness: this check is not necessary, as all information for decryption is carried by the receipt.

5-10 The front end sends receipts to the back-end

11 The back-end processes the receipts to produce intermediate data and cleartext ballots: for a particular receipt $(W_i, \mathcal{D}^i)$ where $\mathcal{D}^i = \{seed_i; \mathcal{D}^{i-1}\}_{K_i}$, and W_i is the encrypted ballot, the back-end decrypts $\mathcal{D}^i$, gets $seed_i$, computes $W_{i-1} = W_i \circ F(seed_i)$ and outputs $(W_{i-1}, \mathcal{D}^{i-1})$ in a permuted order on the SPBB.

12 The back-end proves that the published receipts are equivalent to the cleartext ballots: each mix individually proves that it did the decryption correctly and it transformed the payload W_i correctly (for example, using randomized partial audits).

5.1.2 Advantages and disadvantages

The onion mixnet is one of the oldest mechanisms for anonymous delivery. Because it was not designed especially for voting systems, its main advantage is that it is very general, and it can accommodate all types of ballots. In particular, it can accommodate a ballot that has write-in candidates more efficiently than is possible with other back-ends. The back-end is not responsible for producing any initial information about the ballot style; in this sense the back-end is fully detached from the front-end. For the same reason, the details of the election can be specified after the back-end has been set-up.

The setup steps are very simple and there is no checking before the elections. The path that is followed by each receipt during decryption is decided on the fly (the permutation each mix performs is dynamically generated).

The main advantage is that the back-end is a truly distributed authority. Each mix can operate independently of the other mixes, and no single mix can determine how any voter votes (that is, no single mix can link an input receipt with a cleartext ballot). This can be very attractive in scenarios where the authority running the back-end is distributed geographically, and cannot participate in a secret sharing scheme that requires in-person participation.

Because each mix uses advanced cryptographic techniques to transform the ballots, an onion mixnet may be difficult to explain to the non-technical voter. Further, because a mixnet uses asymmetric encryption, it can be very inefficient. For large elections (millions of cast votes), an onion back-end can take an unacceptably long time to produce a tally (see Sect. 5.4.1).

5.2 Punchscanian mixnets

This dissertation presents the PunchScan back-end, with proofs of integrity and privacy. It also presents variations of the original back-end that address the following issues: the influence of the configuration parameters on integrity and privacy; the tradeoff between them; the generation of partial verifiable tallies; recovery from failed audits; privacy audits; the mitigation of attacks involving voting patterns (including undervotes). Each of these aspects is discussed both informally and formally in separate sections below.

5.2.1 Mathematical model

We describe a model for a single contest. Let V^* be the ordered set of candidates in a particular contest, and let the cardinality of the set be $|V^*| = c - 1$. The order of the candidates in V^* is the order in which they appear on the ballot. Let $V = \emptyset \bigcup V^*$, where $\emptyset$ is a vote for nobody (we assume that V^* does not have an explicit "none of the above" vote).

We assume there are n "virtual ballots" to start with, indexed by the elements of the ballot indexing set. We abuse the notation $\mathbb{Z}_n$ to denote the set of numbers from 0 to $n-1$. Let k be the number of candidates that the voter is allowed to select from V^*. Let $V^k = V \times V \times ... \times V$, the k times cartesian product of V with itself. Conditions will typically be placed on the V^k to obtain $\mathcal{V}$, the set of all possible votes; for example, for a non-cumulative contest, no candidate except $\emptyset$ can appear more than once in a valid vote, and $\mathcal{V} = \{v \in V^k | v_p \neq \emptyset \Rightarrow v_p \neq v_q \ \forall \ p \neq q\}$ where v_p denotes the p^{th} component of v.

Let π_1 and π_2 denote bijective functions on ballot indices, $\pi_1 : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$ and $\pi_2 : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$; π_1 and π_2 are shuffles. The PunchScan back-end may be viewed as operating on n lists of all possibilities a voter may choose (n ballots). For simplicity, we assume there are no constraints on the votes, and that a vote consists of k choices, each from V. The back end performs two operations, D_1 and D_2, each a bijective function, consisting of a bijection (partial decryption) that does not affect null votes, and a shuffle. $D_1 : \mathbb{Z}_n \times \mathbb{Z}_c^k \rightarrow \mathbb{Z}_n \times \mathbb{Z}_c^k$ is a bijective function; $D_1(i, z) = (\pi_1(i), f_i(z))$, for some bijective function f_i, and $f_i(z)_p = \emptyset$ if and only if $z_p = 0$, where the subscript p denotes the p^{th} component of the vector. That is, in the bijection, null votes (denoted 0) are always mapped to 0, this allows us to ignore null votes during the processing. $D_2 : \mathbb{Z}_n \times \mathbb{Z}_c^k \rightarrow \mathbb{Z}_n \times V^k$ is also a bijective function; such that

$D_2(i,z) = (\pi_2(i), g_i(z))$ for some bijective function g_i, and $g_i(z)_p = \emptyset$ if and only if $z_p = 0$

Definition 16 (PunchScan back-end). *$\mathcal{D} = D_2 \circ D_1$ is a PunchScan back-end if and only if:*

1. *π_1 and π_2 are random (or pseudo-random) and independent*
2. *f_i and f_j are independent $\forall i \neq j$*
3. *g_i and g_j are independent $\forall i \neq j$*

Intuitively, the back-end takes an input, modifies it according to the composition of established rules and outputs all the results in a shuffled order, $\pi_1 \circ \pi_2$.

5.2.2 Punchscanian Mixnets as Back-Ends

We now describe how the PunchScan back-end implements the steps that it is responsible for in the protocol described in Chapter 4:

1 The back-end is set-up: the PunchScan back-end secretly computes f_i and g_i $\forall i, j \in \mathbb{Z}_n$ and publishes on the SPBB commitments to the functions: $(i, \text{“}D1\text{”}, \mathfrak{C}(f_i(\mathbb{Z}_c^k)))$, $(i, \text{“}D2\text{”}, \mathfrak{C}(g_i(\mathbb{Z}_c^k)))$ where $f_i(\mathbb{Z}_c^k)$ denotes the ordered set of $f_i(z)$ for all $z \in \mathbb{Z}_c^k$, and $\mathfrak{C}$ denotes a commitment. For each function committed to, the commitments are published in the ascending order of the index i.

2 The back-end gives any necessary information to the front-end: the PunchScan back-end provides $\{(i, g_{\pi(i)} \circ f_i) | \forall i \in \mathbb{Z}_n\}$ to the front-end.

4 Both the back-end and the front-end are checked for correctness: an unpredictable subset $\mathbb{A}$ of indices $\mathbb{Z}_n$ is generated in a public manner. For every $i \in \mathbb{A}$, the back-end publishes on the SPBB i, f_i and $g_{\pi_1(i)}$.

5-10 The front-end sends voted receipts to the back-end.

11 The back-end processes the receipts to produce intermediate data and cleartext ballots : for all the receipts (i, w),the back-end secretly computes $D_1(i,w)$ and $D_2 \circ D_1(i,w)$, sorts both outputs by the first index of the outputs, and publishes on the SPBB all the $D_1(i,w)$ and $D_2 \circ D_1(i,w)$.

12 The back-end proves that the published receipts are equivalent to the cleartext ballots: for all the post-images of D_1 (pre-images of D_2), $(\pi_1(i), w_1)$, an unpredictable choice is made to open exactly one of the commitments: f_i (the pre-image) or $g_{\pi_1(i)}$, (the post-image). The back-end then opens the commitment, and also reveals either the pre-image or post image respectively.

5.2.3 Integrity and Privacy Properties

We now provide formal proofs for the integrity and privacy guarantees of the PunchScan back-end. We first demonstrate the integrity and privacy properties.

Theorem 1 (Audit 1). *Let n be the total number of ballots, and k the number of ballots that are incorrectly formed. If f ballots are chosen at random in step 4, the probability of not detecting any of the k incorrectly formed ballots is smaller than $min[(1-\frac{f}{n})^k, (1-\frac{k}{n})^f]$.*

Proof. The proof can be found in Chapter 10 $\square$

Corollary 1. *Let n be the total number of ballots, and k the number of ballots that are incorrectly formed. If $f = \frac{n}{2}$ ballots are checked in step 4, the probability of not detecting any of the k incorrectly formed ballots is smaller than $(\frac{1}{2})^k$*

Proof. If $x \leq min[y, z]$, then $x \leq y$ and $x \leq z$. In theorem Theorem 1, we take $f = \frac{n}{2}$. It follows that $p \leq (1-\frac{f}{n})^k = (1-\frac{\frac{n}{2}}{n})^k = (1-\frac{1}{2})^k = (\frac{1}{2})^k$ $\square$

As an example, if we assume that 20 ballots are formed incorrectly, then the probability of not detecting any of these when half of the ballots are checked is less than one in a million (regardless of the number of total ballots).

Note that n is not the number of ballots that is going to be used in the election, but the number of total ballots that are initially committed to. The number of ballots that can be used in the election is at most $n - f$.

We now demonstrate that, in fact, the above results imply that the probability of an inconsistent ballot not being detected can be made as small as desired by choosing appropriate values of the total number of ballots, n, and the number of audited ballots, f.

Corollary 2. *Given R, $0 < R < 1$, the values of n and f can be chosen such that $Pr[$an inconsistent ballot is not detected $] \leq R$.*

Proof. The proof can be found in Chapter 10. □

Theorem 2. *[Audit 4] If the back-end does not compute $D_1(i, w)$ or $D_2 \circ D_1(i, w)$ correctly for k ballots in step 11, the probability that all the records on the SPBB are consistent at the end of the protocol is $(\frac{1}{2})^k$.*

Proof. The proof can be found in Chapter 10. □

We now demonstrate the privacy properties.

Theorem 3. *If the commitments are perfectly hiding (perfectly binding) no information is revealed to the computationally unbounded adversary (computationally bounded adversary) by the PunchScan back-end during step 1*

Proof. In step 1 two categories of information could be leaked:

1. Either of the two functions committed to
2. The relationship between the functions

Commitments to the functions do not reveal information on the functions. Further, because the order in which the commitments are published is fixed and predetermined, the relation between the functions is kept hidden □

Theorem 4. *Step 4 does not reveal any information about the unrevealed ballots.*

Proof. Let $j \in \mathbb{Z}_n \setminus \mathbb{A}$. Because $D_1(i, \_)$ and $D_1(j, \_)$ are independent $\forall i \neq j$, then revealing $D_1(i, \_)$ does not reveal any information about $D_1(j, \_)$. Because both π_1 and π_2 are (pseudo-)random permutations, revealing some of the post-images of the functions keeps the unrevealed ones hidden. □

The propositions below assume that none of the ballots is under-voted. Theorem 10 presents an analysis on under-voted ballots.

We will need the following lemma:

Lemma 5. *For any pair of bijections $f : A \to B$ and $h : A \to C$, $\exists$ bijection $g : B \to C$ such that $g \circ f = h$*

Proof. Is is straightforward to check that $g = h \circ f^{-1}$ is such that $g \circ f = h$. f^{-1} always exists because f is a bijection and thus invertible. □

The following are results on the relationship between the privacy of the protocol and the number of pre and post-images checked in step 12.

Lemma 6. *If, in step 12, the unpredictable choice requires that the commitment to f_i be opened for all i, no information is revealed about the correspondence between any receipt (i, w) and $D_2 \circ D_1(j, w)$, $\forall i, j$.*

Proof. Because commitments to only one of f_i and $g_{\pi_1(i)}$ are opened, it follows that no function $D_2(\pi_1(i), \_)$ is revealed. The proof then follows directly from lemma 5. □

The same statement and proof can be made when all g_j are revealed.

Lemma 7. *If, in step 12, the unpredictable choice requires that the commitment to f_i be opened for a single given i, and g_j for all the rest, $j \neq \pi_1(i)$, the receipt with serial number i can be linked to a cleartext vote.*

Proof. $\pi_2(j)$ is revealed $\forall j, j \neq \pi_1(i)$. As π_2 is a permutation, $\pi_2(\pi_1(j))$ is also known. Thus the receipt with serial number i is linked to the cleartext vote at (known) position $\pi_2(\pi_1(j))$ □

In the classical voting literature a variation of the above attack is known as the $n - 1$ attack, where an adversary knows $n - 1$ votes and the tally, and hence knows the n^{th} vote.

Lemma 8. *Suppose, in step 12, the unpredictable choice requires that the commitment to f_i be opened for q values of i, $1 < q < p - 1$, where p is the number of cast ballots. Then the privacy of any of the q corresponding ballots, as defined in definition 5, is q, and that of any of the other ballots is $p - q$.*

Proof. The proof can be found in Chapter 10 □

Theorem 9. *Suppose, in step 12, the unpredictable choice requires that the commitment to f_i be opened for q values of i, $1 < q < p - 1$, where $p \geq 4$ is the number of cast ballots. Then no single encrypted vote $\mathcal{F}(i, v)$ can be traced to a cleartext vote v.*

Proof. From lemma 8, the minimum possible size of the privacy set is 2. If follows directly that, in the worst case, there are at least two cleartext votes that are associated with an encrypted vote $\mathcal{F}(i, v)$. □

The following theorem deals with ballots that are under-voted. We define $UE(UndervotePattern)$ as the set of encrypted ballots $\mathcal{F}(i,v)$ which contain 0 in a certain set of positions, denoted by $UndervotePattern$. We define $CU(UndervotePattern)$ as the set of cleartext (decrypted) ballots that contain 0 in the set of positions denoted by $UndervotePattern$.

Theorem 10. *If the integrity checks all pass, then an element of $CU(UndervotePattern)$ could only have come from an element of $UE(UndervotePattern)$.*

Proof. This is a direct consequence of the fact that both D_1 and D_2 do not change the inputs 0, nor their position. □

In particular, if $|UE(UndervotePattern)| = 1$ for some value of $UndervotePattern$, then that particular receipt can be traced to a cleartext ballot. This result is important as it also reduces the privacy of the other non-under-votes.

5.2.4 Advantages and disadvantages

The unique design of the PunchScan back-end provides a different set of properties from those of traditional onion mixnets. Because the only cryptographic primitive used is a commitment function, and that primitive is only used in step 1 of the protocol, the entire back-end can be explained to an audience that is not familiar with classical cryptographic primitives such as encryption, decryption or digital signature.

The PunchScan back-end is extremely efficient. In practice, millions of encrypted ballots can be transformed to cleartext ballots in minutes using a regular off-the-shelf computer. Sect. 5.4.2 contains a detailed performance analysis of the PunchScan back-end.

All the data used by the PunchScan back-end is deterministically generated. This is needed in order to regenerate the data at subsequent steps of the protocol. The implemented PunchScan back-end offers flexibility in choosing the particular manner in which the data is deterministically generated. For example, f_i and g_j can be simple versions of the shift cipher instead of a substitution cipher, and the commitments can be implemented using any of several standard approaches.

The main disadvantage is the centralized nature of the back-end. The impact of this can be somewhat mitigated by distributing the authority to multiple players using a threshold secret sharing scheme; however, at the exact moment when the back-end data is generated, the machine generating the data can be viewed as a centralized entity. A minor disadvantage is that the details of the election (e.g. number of races and number of candidates per race) need to be known before setting up the back-end.

5.2.5 Generalization

One may generalize the PunchScan back-end to one in which there are many pairs of functions, D_1^j, D_2^j, instead of a single pair D_1 and D_2. In a sense this means creating many such back-ends; however, these back-ends are all configured and used for the same election, and provide enhanced privacy and integrity properties. In this section we discuss introducing pairs of the form D_1^j, D_2^j, each with associated shuffles π_1^j and π_2^j and functions $\{f_i^j\}_{i\in\mathbb{Z}_n}$ and $\{g_i^j\}_{i\in\mathbb{Z}_n}$, in Sect. 5.2.6 we discuss their influence.

First, we consider the case when all the pairs D_1^j, D_2^j map the same input to the same output.

Definition 17 (PunchScan back-end with multiple linked `decryption tables`). *Let* $\mathcal{D}^1$ *be a PunchScan back-end according to definition 16. We define a* PunchScan back-end with multiple linked `decryption tables as a set of PunchScan back-ends` $\{\mathcal{D}^j\}_{j=1}^d$`,` $d > 1$`, each with associated shuffles` π_1^j `and` π_2^j`, and functions` $\{f_i^j\}_{i\in\mathbb{Z}_n}$ `and` $\{g_i^j\}_{i\in\mathbb{Z}_n}$`, such that` $\mathcal{D}^j(i,w) = \mathcal{D}^l(i,w)$`,` $\forall l, j, i, w$`.`

According to definition 17, each function $\mathcal{D}^j$ maps the same input to the same output; in particular, each receipt is mapped to the same cleartext in the same position.

Definition 18 (PunchScan back-end with multiple unlinked `decryption tabless).` *Let* $\mathcal{D}^1$ *be a PunchScan back-end according to definition 16. We define a* PunchScan back-end with multiple unlinked `decryption tables as a set of PunchScan back-ends` $\{\mathcal{D}^i\}_{i=1}^d$`, with` $d > 1$`, such that,` $\forall\ j \in \mathbb{Z}_n$`,` $\forall\ i > 1$`,` $\exists l$ `such that` $\mathcal{D}^i(l,w) = \mathcal{D}^1(j,w)$`,` $\forall l, j, i, w$`.`

According to definition 18, each function $\mathcal{D}^i$ produces essentially the same output in terms of decrypted votes, but ordered differently for each possible back-end.

With this simple modification to the original PunchScan back-end, we now modify the steps of the protocol so as to use all the $\mathcal{D}^i$. A number of possibilities can be considered:

- Row-Wise Commitments: for each $\mathcal{D}^i$, in step 12, and for each j, either $D_1^i(\pi_1^{-1}(j), \_)$ or $D_2^i(j, \_)$ is revealed, but never both, as before. In other words the choice of which partial transformation to audit is different for each ballot, for each $\mathcal{D}^i$

- Column-Wise Commitments: for each $\mathcal{D}^i$, in step 12, either the entire D_1^i or D_2^i is revealed, but never both. In other words, for each set of transformations $\mathcal{D}^i$, the choice of which partial transformation to audit is the same for all the ballots.

If the second option is chosen, then, in step 1, there is no need to compute separate commitments to $D_1^i(j, \_)$ and $D_2^i(\pi_1(j), \_)$. Instead, commitments to the pair of functions $(D_1^i(j, \_), D_2^i(\pi_1(j), \_))$ are computed and published separately for all (i, j). Also published is a separate commitment to the function D_1^i and the function D_2^i for all i. This reduces the number of commitments to approximately half, and thus improves the efficiency of this particular operation by a factor of 2.

In the next section we discuss the influence of these two choices for the integrity and privacy guarantees offered by the back-end.

5.2.6 The influence of the configuration parameters on integrity and privacy

We discuss the influence of the value k on the integrity and privacy offered. Based on our findings, we make recommendations for parameters that should be used for various types of elections.

There are four possible cases: row-wise commitments with linked d͡ecryption tables, row-wise commitments with unlinked d͡ecryption tables, column-wise commitments with linked d͡ecryption tables and column-wise commitments with unlinked d͡ecryption tables.

Since step 1 of the protocol does not have any influence on the privacy of the system, as proven by Theorem 4, and the probability of not detecting that cheating occurred can be made arbitrarily small, as proven by corollary 2, we focus our attention on step 12 of the protocol and see what impact it has on the integrity and privacy.

We consider an election with n cast ballots. In our analysis below, we observe a trade-off between integrity and privacy. We assume that all the ballots have been fully voted (do not contain under-votes).

Theorem 11. *For a PunchScan back-end with d linked or unlinked $\overset{\frown}{d}$ecryption tables, row-wise commitments and n cast ballots, assume that there exist k ballots for which the back-end did not compute $D_2 \circ D_1$ correctly. Then the probability that the records on the SPBB are consistent at the end of the protocol is $(\frac{1}{2^k})^d$.*

Proof. The proof can be found in Chapter 10. □

Theorem 12. *For a PunchScan back-end with d linked or unlinked $\overset{\frown}{d}$ecryption tables, column-wise commitments and n cast ballots, assume that there exist k ballots for which the back-end did not compute $D_2 \circ D_1$ correctly. Then the probability that the records on the SPBB are consistent at the end of the protocol is $(\frac{1}{2})^d$.*

Proof. The proof can be found in Chapter 10. □

Theorem 12 is interesting: the integrity assurance does not depend on the number of ballots cheated on, but only on a parameter that can be fixed before the election. Moreover, for any given integrity assurance level, one can choose the parameter d such that the integrity level is guaranteed. For example, if we want to run an election with 99.9% assurance that the result was correctly computed from the published data, then we take $d = 10$ ($1 - \frac{1}{2^{10}} \approx 99.9\%$).

The result stated in Theorem 12 can also be read as follows: it is as easy to cheat on all the ballots as it is to cheat on one ballot. This is an immediate consequence of the fact that the integrity assurance does not depend on the number of ballots cheated on.

Theorem 13. *For a PunchScan back-end with d linked or unlinked $\overset{\frown}{d}$ecryption tables, column-wise commitments and n cast ballots, the privacy of any of the ballots at the end of the protocol is n.*

Proof. The proof can be found in Chapter 10. □

We now make an analysis of the privacy for linked and unlinked $\overset{\frown}{d}$ecryption tables If audits are performed as described, it is difficult to make a mathematical model for the privacy offered by the PunchScan back-end when it is set-up to use linked or unlinked $\overset{\frown}{d}$ecryption tables. We start by giving a few examples and we discuss informally the privacy properties offered by such setups.

First we assume that the PunchScan back-end is set-up with linked $\overset{\frown}{d}$ecryption tables. This means that $\mathcal{D}^i(l, w) = \mathcal{D}^j(l, w)$, i.e. the order of the outputs for all of the $\mathcal{D}s$ is the exact same. Assume that, for each integrity audit, half of the D_1^j transformations are fully opened, and, implicitly, the other half of D_2^j is also opened. Let I_1^j be the set of preimages of the D_1^j that is opened, I_2^j be the rest of the preimages of D_1^j. We can infer that I_2^j is the input that maps to the output fully revealed by D_2^j, because it could have not corresponded to any of the unrevealed outputs, so it must be corresponding to one of the unrevealed inputs. Let $O_1^j = \mathcal{D}(I_1^j)$ and $O_2^j = \mathcal{D}(I_2^j)$.

We consider an example, with eight ballots, each ballot having two candidates, A and B. The final tally is four votes for A and four votes for B. The input for any $\mathcal{D}$ is

$$(0, w_0), (1, w_0), (2, w_2), (3, w_3), (4, w_4), (5, w_5), (6, w_6), (7, w_7)$$

and the output is

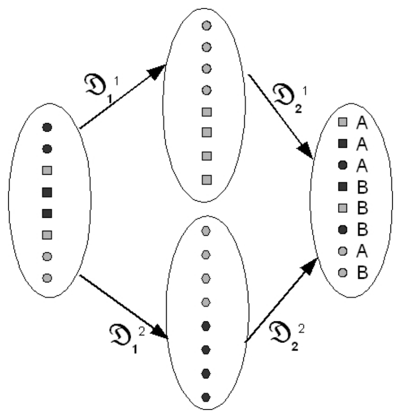
$$(0, A), (1, A), (2, A), (3, B), (4, B), (5, B), (6, A), (7, B).$$

Also, assume that there are two $\mathcal{D}$s.

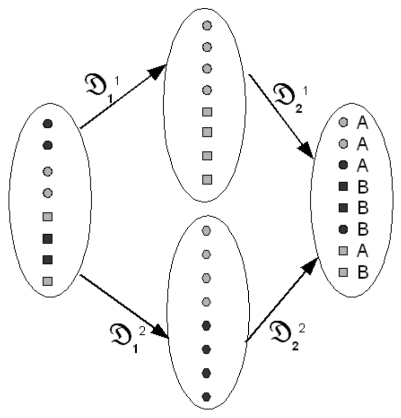
We now consider some of the various configurations that are possible for the integrity challenges. We first present a best case scenario, we continue with a worse scenario and we end with the worst case scenario.

Before the integrity audit, when examining the outputs of the PunchScan backend in our example, the probability that any of the inputs maps to a cleartext vote for a candidate is maximal. In the ideal case this is equal to the probability given by the percentage of the votes each candidate got, which is public. In our example, each candidate got four votes, so the probability is 50%. The best case scenario is when, after the integrity audit, this probability stays the same.

Assume that $I_1^0 = (0, 1, 6, 7)$ and $O_1^0 = (2, 5, 6, 7)$ and thus $I_2^0 = (2, 3, 4, 5)$ and $O_2^0 = (0, 1, 3, 4)$. In this particular example (see Fig. 5.1(a)) O_1^0 has the sets of all possible clear votes (A, B, A, B), in the same percentages as the total tally. Therefore this particular integrity check does not affect privacy at all. Furthermore, assume that another integrity check is made for the second $\mathcal{D}$, $I_1^1 = (0, 1, 3, 4)$ and $O_1^1 = (1, 2, 3, 5)$ and thus $I_2^1 = (2, 5, 6, 7)$ and $O_2^1 = (0, 4, 6, 7)$; the same discussion applies for this set of audit. If we combine the two sets of audits we can deduct some sets that have lower cardinality: $I_1^0 \bigcap I_1^1 = (0, 1)$ and $O_1^0 \bigcap O_1^1 = (2, 5)$ and thus the receipts $(0, 1)$ correspond to the clear votes $(2, 5)$. It so happens that the clear votes at positions $(2, 5)$ are (A, B), and thus we cannot distinguish between the two encrypted voted $I_1^0 \bigcap I_1^1$. Moreover, the percentage in this smaller set is the same at the percentage in



(a) Best case scenario for auditing: each of the four sets that resulted from the output set contain a vote for A and one for B



(b) Intermediary case scenario for auditing: only two of the four sets that resulted from the output set contain a vote for A and one for B. The pother two contain a vote for the same person, thus complectly breaking the privacy of four of the eight votes.

Figure 5.1: Various auditing possibilities. The first (top) audit splits the input and the output sets in half: squares and circles. The second (bottom) audit splits the resulting sets in half again: colored or not colored. The result is four sets in the input that correspond to four sets in the output: colored circles, uncolored circles, colored squares, uncolored squares

the final tally, thus no information is leaked. We can also compute $I_1^0 \bigcap I_2^1 = (6,7)$, $O_1^0 \bigcap O_2^1 = (6,7)[A,B]$, $I_2^0 \bigcap I_1^1 = (3,4)$, $O_2^0 \bigcap O_1^1 = (1,3)[A,B]$, $I_2^0 \bigcap I_2^1 = (2,5)$, $O_2^0 \bigcap O_2^1 = (0,4)[A,B]$. In all cases, no more information can be inferred from having the second $\mathcal{D}$ audited. While it seems that no privacy is lost, now there are four smaller tallies created, and other attacks can now be conducted on these tallies. In particular an $n-1$ attack can be conducted on any of the smaller tallies, as follows: assume that Alice is a voter that got receipt number 2; then Alice knows how receipt number 5 has voted, because it is now known that receipts 2 and 5 correspond to (A,B).

We now consider another scenario (see Fig. 5.1(b)), where it so happened that $I_1^0 = (0,1,2,3)$ and $O_1^0 = (0,1,2,5)$ and thus $I_2^0 = (4,5,6,7)$ and $O_2^0 = (3,4,6,7)$. In this case O_1^0 contains the cleartext votes (A,A,A,B), and thus the distribution of votes is chanced compared to the whole tally: in this partition 75% of the votes are for A and 25% for B, whereas in the whole tally the corresponding numbers are both 50%. Thus the audit tells us that it is more likely for the receipts $(0,1,2,3)$ to correspond to a vote for A than to a vote for B; this was not known before the audit. Furthermore,

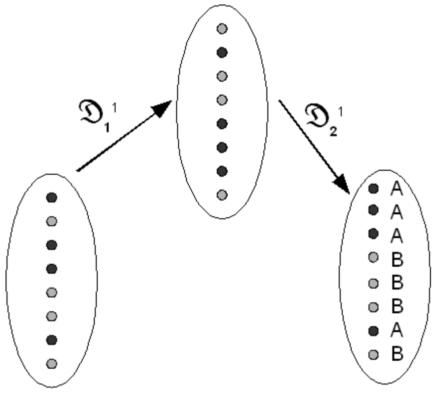


Figure 5.2: Worst case scenario for auditing: even with a single $\mathcal{D}$ the output set of cleartext votes happened to be portioned in two sets such that any of the two sets contain a single type of vote, thus complectly breaking the privacy of all eight votes.

assume that another integrity check is made for the second $\mathcal{D}$, $I_1^1 = (0,1,4,5)$ and $O_1^1 = (2,3,4,5)$ and thus $I_2^1 = (2,3,6,7)$ and $O_2^1 = (0,1,6,7)$; the same discussion applies for this set of audit, but now any receipt in I_1^1 is more likely to correspond to a vote for B, because O_1^1 corresponds to cleartext votes (A, B, B, B). Applying the same rational as in the previous analysis, we compute $I_1^0 \bigcap I_2^1 = (2,3)$ and $O_1^0 \bigcap O_2^1 = (0,1)$; but now we observe that both cleartext votes $(0,1)$ correspond to votes for A, and thus both receipts $(2,3)$ correspond to votes for A. If we consider $I_2^0 \bigcap I_1^1 = (4,5)$ and $O_2^0 \bigcap O_1^1 = (3,4)$, we observe that both outputs 3 and 4 correspond to votes for B, thus the receipts 4 and 5 correspond to votes for B. We have thus violated the secrecy of four ballots in all.

The worst case scenario (see Fig. 5.2) is when $I_1^0 = (0,2,3,6)$, and $O_1^0 = (0,1,2,6)$. All four votes are cleartext votes for A. Similarly, $O_2^0 = (3,4,5,7)$, all cleartext votes for B. Thus, even though a single $\mathcal{D}$ was used to check the correctness, it was possible to determine a vote. It does not matter that it is not known to which particular cleartext vote a particular receipt corresponds to, as long as all the possible cleartext votes are for the same candidate.

In Sect. 5.2.7 we describe a way to safely audit the PunchScan back-end, for both linked and unlinked decryption tables, essentially enforcing the best case scenario.

One may think of the problem as follows. The row-wise commitments and the row-wise auditing partition the set of cleartext votes. But if the distribution of the candidates in the subsets forming the partition is not the same as the distribution on the entire set of votes, this leaks information about the receipts in the particular subset.

This tradeoff can be translated into one between integrity and performance, by simply using column-wise commitments, as is clear from Theorem 12 and Theorem 13. The privacy does not depend on the number of $\mathcal{D}s$. Increasing d decreases the performance of the back-end, resulting in a compromise between integrity and performance.

For large elections (millions of voters), we recommend using the row-wise commitments, unlinked $\widehat{d}$ecryption tables and keeping d low (between 1 and 3), so privacy degradation is low and the integrity is extremely high, for even a small percent margin. Even a 0.01% margin would mean more than a 100 vote margin, which would translate into having to cheat on at least 50 ballots. This would translate into an integrity assurance of at least $1 - (\frac{1}{2^{50}})$.

For a small election (tens or hundreds of voters), we recommend using column-wise commitments with d between 20 and 30. This would ensure a maximal level of privacy and a sufficient level of integrity $(\frac{1}{2})^d$. For example if $d = 20$, this would translate into an integrity assurance of $1 - (\frac{1}{2^{20}}) \approx 99.9999\%$.

5.2.7 Safely auditing the PunchScan back-end

As we have seen in the previous section, auditing the PunchScan back-end at random may reveal information on individual votes. In the worst case, all receipts are linked with certainty to cleartext votes, even when a single $\mathcal{D}$ is used. In this section we describe a way to choose what ballots are audited so as to maximize the size of the privacy sets. This technique ensures that the best case scenario presented in the previous section always happens.

To keep the explanation simple, assume we have an election with only two candidates, A and B, the final tally is 50% for A and 50% for B, and that the number of ballots is a power of two. Instead of making a random challenge for each receipt, the auditor groups the cleartext votes in two sets, such that the distribution of votes in each of the sets is the same as the distribution of the entire set of votes (in our example 50/50). The auditor makes a random choice for one of the sets, between opening g_i or f_i. The choice for the other set is then fixed.

When it comes to auditing a second instance of a table, the auditor now obtains a partition of size two of each of the subsets of the first partition. Again, the distribution of votes in each of the four subsets in the partition is the same as the distribution of the entire set of votes. The cleartext votes from the four smaller tallies are combined by the auditor such that they do not result in an arrangement that was obtained before, and a new challenge is made for the new partitioning. This technique can be applied $log(n) - 1$ times, and thus for $log(n) - 1$ $\mathcal{D}$s, where n is the number of ballots, without revealing a link from the receipts to cleartext votes.

For a better understanding, we show how this technique applies to the example in the previous section. The input for any $\mathcal{D}$ is

$$(0, w_0), (1, w_0), (2, w_2), (3, w_3), (4, w_4), (5, w_5), (6, w_6), (7, w_7)$$

and the output is

$$(0, A), (1, A), (2, A), (3, B), (4, B), (5, B), (6, A), (7, B).$$

Also, assume that there are two $\mathcal{D}$s.

To challenge $\mathcal{D}^0$, the auditor inspects the output and splits it into, for example, $O_1^0 = (0, 1, 3, 4)$ and $O_2^0(2, 5, 6, 7)$. This choice partitions the output cleartext votes into two subsets, each with the same distribution as the larger set of votes. The auditor now makes a choice to see the pre-images of D_2 for either O_1^0 or O_2^0. This choice does not impact the privacy since both of them will produce $I_1^0 = (2, 3, 4, 5)$ and $I_2^0 = (0, 1, 6, 7)$. When analyzing these two partitions, the probability of each of the inputs going to a particular cleartext vote is the same as when analyzing the initial tally.

When auditing $\mathcal{D}^1$, the auditor inspects O_1^0 and breaks it into two, such that the probability distribution of the cleartext votes does not change in the two smaller sets. One way of breaking it is $(0, 3), (1, 4)$. The same operation is done for O_2^0, resulting in, for example, $(2, 5), (6, 7)$. The four smaller sets are now combined into two larger sets, for example $O_1^1 = (0, 2, 3, 5)$ and $O_2^1 = (1, 4, 6, 7)$, and a random challenge is made, as before. The resulting input sets are $I_1^1 = (0, 1, 2, 4)$ and $I_2^1 = (3, 5, 6, 7)$. If we combine this result with the previous results, we can compute the intersection of the input sets and the intersection of the corresponding output sets and get $(2, 4) \rightarrow (0, 3) = (A, B)$, $(6, 7) \rightarrow (6, 7) = (A, B)$, $(3, 5) \rightarrow (1, 4)(A, B)$ and $(0, 1) \rightarrow (2, 5) = (A, B)$, getting the best case scenario described in the previous section.

We now analyze the impact of this auditing method on the integrity. Without this technique, each output could have been chosen as a post-image or not, with probability 50%, and the choice for the next ballot would be independent of the choice for the first ballot. With the technique described here, the second choice is no longer independent of the first choice, since the probability distribution changes when the first ballot is taken out the tally, and the distribution in the final tally has to be maintained. For example, if two outputs that correspond to the candidate choice of A have already been grouped together, the other two with candidate choice A cannot be opened on the same side (D_1 or D_2), as this would change the distribution of votes in the subsets of the partition.

In the default case, there are 2^n possibilities for partitioning the input. We now compute the number of ways we can partition the output using the described technique. For simplicity, we still assume a two candidate race and a tie. The number of ways the audit on the first $\mathcal{D}$ can be conducted is the number of possible ways of breaking the output set into two sets of equal size that have the same distribution of cleartext votes. Assuming there are n ballots, $\frac{n}{2}$ for A, and $\frac{n}{2}$ for B, each set needs to have $\frac{n}{4}$ votes for A, and $\frac{n}{4}$ for B. The number of ways of choosing $\frac{n}{4}$ votes for A out of $\frac{n}{2}$ is ($\frac{n}{2}$ choose $\frac{n}{4}$)=$\begin{pmatrix} \frac{n}{2} \\ \frac{n}{4} \end{pmatrix}$

We now generalize this observation and prove that this new method provides essentially the same level of integrity, while guaranteeing maximal privacy, whenever possible.

Theorem 14. *Assume in an election there are c candidates and each candidate receives n_i votes, $\sum_{i=1}^{c} n_i = n$. The number of ways to partition the set of votes into two subsets each with the same distribution of votes, and to choose one of the tallies, is* $\frac{2^n}{\sqrt{\prod_{i=1}^{c} n_i}}$

Proof. The proof can be found in Chapter 10. □

For all practical cases, the number offered by Theorem 14 is as good as 2^n, the number of possibilities without the technique presented in this paper. If RPC divides the set $\mathbb{P}$ into two sets with the same cardinality, the integrity offered is $2 \times \begin{pmatrix} \frac{n}{2} \\ n \end{pmatrix} \approx 2 \times \frac{2^{n-1}}{\sqrt{n}}$[1], a number even closer to $\frac{2^n}{\sqrt{\prod_{i=1}^{c} n_i}}$.

For example, if $n = 10000$, $c = 10$, and all n_is are equal to $\frac{n}{c} = 1000$, the number of possibilities is $\frac{2^{10000}}{\sqrt{1000^{10}}}$=$2^{10000} \times \frac{1}{1000^5} \approx 2^{10000} \times \frac{1}{(2^{10})^5}$=$2^{10000-50}$=$2^{9950}$ a number very close[2] to 2^{10000}, the ideal number of combinations, and even closer to $2^{10000} \times \frac{1}{\sqrt{10000}} \approx 2^{9993}$, the number of choices offered by RPC when the cardinality of the two sets is equal.

[1]We assume RPC divided the set on which it does the random choices in exactly half, and chooses which way to open one of the two halves, thus the formula

[2]actually $\frac{1}{2^{9950}}$ is very close to $\frac{1}{2^{10000}}$

Note that in some cases, our technique may be unable to keep the exact same distribution of messages; if the number of outputs carrying the same message is odd, one cannot divide it exactly in half. In particular, if there is a single output carrying a unique message (a single vote for one of the candidates), our technique will result in revealing that half of the inputs do not correspond to that message. The original RPC technique suffers from the same problem.

5.2.8 Recovering from failed audits

One of the worst scenarios that an election authority can imagine is to start an election, collect encrypted ballots, decrypt them and publish the results (steps 5 to 10 of the general protocol) and in step 12 the data published by the back-end is not consistent with all the other data already published on the SPBB. On the one hand, this should never happen; the software should have been tested ahead of time, there should be no intentional cheating. etc. Nevertheless this is a possible scenario, and requires a recovery plan. Another scenario requiring a recovery plan is one where the entire process goes well, but one of the losers of the election successfully convinces a judge that the election authority needs to bring additional evidence related to the integrity of the election. A third scenario is one where the voters complain and successfully prove that the SPBB contains some data that is not consistent with the voter receipts; but this complaint comes in after the complaint period, and after the results have been posted and audited. Such voters may be able to convince a judge that the election was rigged, regardless of the fact that the complaints came in too late. We describe a very simple way in which the back-end can prepare for such situations. If the back-end takes preventive measures and is prepared for such scenarios, a failed audit does not automatically imply a re-run of the election, or the compromise of voter privacy by the full opening of all the commitments published in step 1.

Assume that d is the number of $\mathcal{D}s$ the back-end started with. Then, in step 12, a random set of $\mathcal{D}s$ is chosen and the random choices are made only on this particular subset. The remaining $\mathcal{D}s$ are kept hidden. These $\mathcal{D}s$ will be used in any subsequent dispute resolution process: step 11 and step 12 will be repeated and will presumably solve the problem that triggered this additional integrity check. These are new proofs of integrity that are independent of any previous proofs, since the $\mathcal{D}s$ are independent among them.

It is essential that the back-end have enough $\mathcal{D}s$ such that the integrity guarantees provided each time the last steps are repeated are sufficient to convince the authority that required the additional proofs. Thus the back-end should prepare a larger number of $\mathcal{D}s$ in the initial first step.

Note that it is not possible that the back-end generate a fresh set of $\mathcal{D}s$ when additional evidence is required. The reason is that the back-end already knows the encrypted votes and could generate $\mathcal{D}s$ that do not truly recover the cleartext votes from the encrypted votes. The back-end can generate a set of $\mathcal{D}s$ that would transform any the encrypted votes to any desired cleartext votes.

5.2.9 Contest partitioning

The ability to trace a ballot to the voter, through the examination of information on the ballot, is a well known risk in traditional voting systems. Since all of these techniques require the cooperation of the voters themselves, they are mainly a concern because of the possibility of vote-buying and voter-coercion. Consider, for example, elections using hand-counted paper ballots. If a voter can write-in an arbitrary candidate name for a certain office, they could write in their own name for that office, or a pre-determined keyword if their vote was being purchased. Then, during the count of ballots, anyone who can view the ballots (officials, observers, etc) can see the voter's name or the keyword and verify that they voted as instructed by the coercer. Using write-in candidates to uniquely identify ballots is a problem in any voting system which allows the write-in marks to be coupled with the rest of the ballot. However, even if write-in votes are prohibited or completely separated from the main ballot, forms of ballot tracing are still possible and are the main focus of this section. Since write-in votes can easily be separated from the rest of a ballot (whether on paper or in electronic form) and processed separately (in the same way that absentee ballots are processed), or write-in votes can be restricted to only pre-approved *write-in candidates*, this specific form of vote-buying has possible solutions independent of any particular voting system.

In a slightly more complicated, but less obvious, method, the voter may mark a specific pattern of votes, say for an office of relatively less importance (for example "Town Dog Catcher") but which has several candidates running, while voting for the instructed candidate for the major contests. Again, someone who can observe the votes, can recognize the pattern of votes made in the "Dog Catcher" contest, and know that this is a purchased or coerced vote and verify that the major contests are voted as instructed. Clearly this technique provides less information/identity leakage than the write-in vote, as there would be only about 5-10 possible vote patterns for a typical contest.

For example, Alice runs in the first contest (and is candidate number one). She has coerced Victor, Valentino and Valerie to vote for her for contest one, and instructed them to vote for the third, second and first candidates in contests two, three and four respectively. After the election is complete, Alice can look at the output of $\mathcal{D}$ and see that there are three ballots that have those exact vote patterns. She also sees that there is a vote for her in each of those three ballots. Thus she can verify that the coerced voters cooperated. However, if she only finds two ballots in the output of the back-end table with the correct pattern, she will not know which of the three, Victor, Valentine or Valerie did not comply.

A second vulnerability exists because of the way in which undervotes are handled. In almost all elections, voters are allowed to "not vote" for any particular office – and in many of those cases they do not have to mark something on the ballot to indicate they are not voting for that office, they can just leave it blank. This is called an *undervote* as the voter did not vote as many times (or for as many offices) as they were permitted. The encrypted votes, the input to the D_1 function, are made public and the undervotes are visible in the clear. Because of the way the PunchScan back-end handles undervotes, they will be translated unmodified by the D_1 and D_2 functions ($D_1(i,z)_{p+1} = 0$ and $D_2(i,z)_{p+1} = 0$ if and only if $z_p = 0$) and they appear in the exact same position and thus in the exact same pattern in the output that contains cleartext votes.

For example, Alice again tries to buy the election. She is running in the first contest as candidate number one. She has coerced three voters to vote for her and she told Victor to abstain in contests 2 and 3, Valentino to abstain in contest 2 and 4 and Valerie to abstain in contests 3 and 4. After the election, Alice can look at the output of D_2 and see that there are three ballots that have those exact positions undervoted. She also sees that there is a vote for her in each of those three ballots. This attack is even more powerful because Alice can link these ballots from the output of $\mathcal{D}$ to its input, which retains the serial number of her receipt. Thus if a coerced voter does not mark the required undervote pattern, their serial number can be determined and then Alice can ask each coerced voter to show their receipt, matching serial numbers and the requested undervote pattern. This allows Alice to verify who did not vote "correctly" when multiple voters were assigned the same pattern. In practice the serial number is often linkable to the voter's identity, for example because the poll-workers see both, or the same computer systems process both registrations and ballot scanning or generation. Whether or not these two are linked does not affect the existence of the attacks.

Thus, the best way for a coerced voter to avoid voting as instructed, is for them to vote the undervote pattern, but actually vote for someone else in the first contest. As long as there is at least one other person voting the same undervote pattern, they cannot be distinguished. How useful this is depends on how many other voters are assigned the same pattern.

To measure how dangerous this undervote attack actually is, one must consider the ability of the attacker to accurately trace individuals to their coerced votes. If only a few undervote patterns are possible, the coercer has a limited choice of either only coercing a few votes, which won't really affect the election, or coercing a lot of votes but having no way to verify or enforce each one. So to analyze the power the attacker has, we need to minimize the number of undervote patterns possible.

As a result, on a ballot with only a few offices, for example many European elections with only members of parliament or maybe a local mayor on the ballot, this risk is not substantial as each office only allows one bit of information to be leaked (did you undervote for office X or not), giving only a few undervote patterns. However, with ballots containing many contests and issues, such as in the United States, the amount of information that can be leaked becomes meaningful.

We define a measure U which represents the number of possible ways to fingerprint a ballot using undervotes. On a ballot with q contests, and with q_i possible candidates' choices for each contest:

$$U = \prod_{i=1}^{i=q} 2^{q_i}$$

If only one candidate can be chosen for each contest, $U = 2^q$, thus if $q = 12$ and there are 4092 ballots in total, all of them can be uniquely fingerprinted and traced. This would ruin privacy for all voters who undervote in a known pattern. In practice, fewer fingerprints will actually be possible for a given number of contests, as in vote-buying or coercion some of the contests need to actually be voted for. So some of the contests can not be undervoted to form a fingerprint. But one can easily imagine that 2 or 3 top-of-the ballot contests could be worth vote buying even if it forced the rest of the contests to be voted in a specified pattern of undervotes. This would only decrease the number of fingerprints to 512 or 1024, still enough to be considered a serious attack.

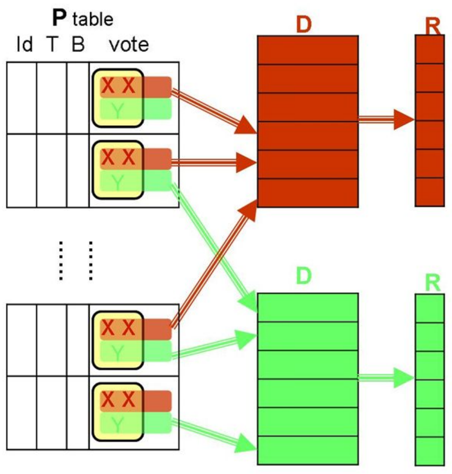


Figure 5.3: Partitioning a ballot into red and green questions which are processed separately

We describe a simple way of preventing the fingerprinting of ballots through the use of undervote patterns. The key idea is process separately the votes for each contest. This unlinking will mean that the set of votes for contest 1 (say President) can not be associated with the same voter's votes for other contests such as Senator. This type of unlinking is quite unlike traditional paper ballots, or optical scan systems, where the physical ballot is kept as a single, unified piece of paper during the vote counting, recount, and post-election processing. So how a voter voted on contest 1 is clearly linked to how he or she voted on contest 2. Other voting technologies such as DRE or Mechanical Lever machines do not have this property of linkability of contests as they do not record a 'ballot' in the traditional sense, but only store the tallies of votes for each office. However, while preventing the possibility of the contest linking, these technologies also prevent meaningful recounts or ballot auditing. PunchScan can unlink the contests while preserving the strong auditability of cryptographic and paper ballot systems.

To unlink contests in PunchScan, we conceptually partition the entire set of contests on a ballot into several subsets, and use a $\mathcal{D}$ back-end for each subset. All $\widehat{d}$ecryption tables are unlinked. For example, in Fig. 5.3 we show a simple example of a ballot with two subsets, one labeled in red and one labeled in green. The full submitted ballot is stored as the input to $\mathcal{D}$, however, each portion of the ballot that corresponds to a particular subset is linked to a different $\mathcal{D}$, so the red responses can no longer be linked to the green answers when they reach their respective outputs, as they are mixed independently. At the extreme, each subset contains only one contest.

For a single section, the number of ways to fingerprint a ballot is:

$$U = \prod_{i=1}^{i=s} 2^{q_i}$$

where s is the number of questions in the section. Analyzing the maximum number of candidates that can be selected for each contest, the ballot can be partitioned in sections such that the value of U does not get too high for any section.

The use of separate $\mathcal{D}$ for each section results in the original ballot, with all of the response marks on it, being split into separate groups of contests and each contest group being decrypted into a cleartext vote through its own $\mathcal{D}$. Since the cleartext votes for each contest are now stored separately and can not be linked to the cleartext votes for other contests that were made by the same voter.

No information about other contest votes is released since the linkage between those contest results and the original ballots are protected by a separate $\mathcal{D}$ with its own commitments and permuted relationships. The integrity of the election is not weakened, as this technique only separates contests within the ballot, so the same audit procedure is employed and the same risk of exposure exists for malicious election authorities.

Contests cannot be unlinked arbitrarily, as sometimes there are cases where contests are naturally coupled. For example, in an impeachment vote, it may be required that in order to vote on who should replace the current office holder, you must also vote to impeach the current office holder (whether such a rule is good public policy or not is a question we do not discuss). So with the two contests:

1. Do you want to impeach the mayor?

2. Who do you want as new mayor if the old mayor is impeached?

a voter can answer the second question only if he answered “Yes” to the first one. This type of coupling cannot be enforced if the contests are processed separately. Coupled contests must belong to the same subset of contests.

Information in coupled contests may be revealed in other ways through undervotes. For example, just the existence of a mark for the second contest indicates that the voter probably chose “Yes” to question 1. Correspondingly, if there is no mark for the second contest, the most likely answer to the first is “No”. These tight relationships between contests cannot be fixed by the technical changes we propose here, but can most easily be fixed in the design of the ballot by having an explicit “no” vote for the second contest.

A special case of coupled contests is the ability to cast straight-party ballots; all contests related to straight-party ballots must appear in the same partition to avoid double-voting.

A slightly different way of fingerprinting the ballot is to vote for a certain pattern of candidates in contests that may be considered unimportant. These patterns may uniquely identify a ballot and thus also reveal the votes for all the contests. The current proposal also solves this problem by unlinking the contests on a single ballot.

This solution does not affect the integrity or the overall privacy of the election. The integrity in some contests may be considered more important then in others, thus a greater number of tables can be used.

5.2.10 Information theoretic integrity vs. information theoretic privacy

In this section we investigate the consequences of using the three types of commitments (perfectly hiding, perfectly binding or neither) in the PunchScan back-end. Recall that a commitment scheme cannot be both perfectly hiding and perfectly binding.

The very first step of the PunchScan back-end protocol is to publish commitments. Moreover, commitments are the only cryptographic function used by the entire PunchScan back-end protocol; there is no classical encryption or decryption using symmetric or asymmetric keys and no signing of message is strictly required. The security of the PunchScan back-end depends on, first, the soundness of the protocol, with has been proven by the theorems in Sect. 5.2, and second, on the security guarantees offered by the commitment scheme used.

An intuitive result is that, if information theoretic integrity is desired, it is necessary to use a commitment scheme that is perfectly binding. The reasoning is simple: even with unbounded computational power, the back-end should not be able to open a commitment in two distinct ways, because this would affect the integrity offered by the back-end.

Theorem 15. *A PunchScan back-end has information theoretical integrity if and only if it uses a perfectly binding commitment scheme.*

Proof. The proof can be found in Chapter 10. □

We now look at what guarantees can be provided if a perfectly hiding commitment scheme is used. A perfectly hiding commitment scheme ensures that a computationally unbounded adversary does not learn any information from the commitment itself. It is tempting to say that privacy is assured even for a computationally unbounded adversary, but this depends on a number of assumptions. First, one has to assume that the technique described in Sect. 5.2.9 is used, and that patterns of votes inside a single contest are not possible. Second, the $n-1$ attack described in lemma 7 is not possible. Third, Theorem 9 proves that the privacy is not optimal, i.e. it is not at its largest (the total number of cast ballots).

5.2.11 Conclusions

We have formally presented the PunchScan back-end, using a mathematical model. We have computed the integrity assurances that are provided at each step; we described techniques that allow integrity assurance to be traded-of for perfect privacy and lower performance, and techniques that allow for the privacy set to be maintained at the maximum possible while still allowing for a probability of cheating successfully decreases exponentially with the number of votes cheated upon. We have described general techniques for protecting privacy, including how to generally avoid the Italian attack. Finally we have described practical solutions for how to recover from failed audits and have presented the influence of the commitment scheme used on the integrity and privacy guarantees offered by the PunchScan back-end. Sect. 5.4.2 contains performance analysis.

5.3 Pointer mixnets

We present a simplification of the PunchScan mixnet that takes the contest partitioning idea presented in Sect. 5.2.9 to an extreme: each partition contains only one candidate; thus each encrypted vote has its own independent path that it follows through the mixnet. This type of mixnet is a general construction for an anonymizing mechanism of a fixed set of messages c that can come from n sources; it is very simple to explain to people that do not know anything about cryptography and it also has very interesting theoretic properties. An intuitive description is provided in the Appendix (see Chapter 11), a more formal description is provided in this section.

5.3.1 Mathematical model

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).

We first present a description based on tables, and then present a formal description. 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 votes; T (stands for tallies and results) contains cleartext 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). Fig. 5.4 gives an example of the three tables for an election with six ballots and two candidates.

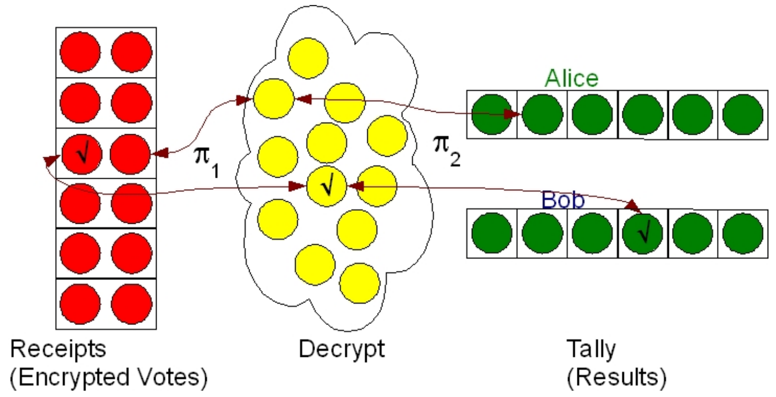


Figure 5.4: Pointer-based mixnet

The tables are connected by two permutations, π_1 and π_2. π_1 connects R (receipts; coded votes) with D (decrypt): $R_k = D_{\pi_1(k)}$, where k is some canonical representation of (i, j); for example, $k = (c-1)i + j$. π_2 similarly connects D with T (results or tally): $D_k = T_{\pi_2(k)}$. These permutations are constrained to map a mark for a particular candidate in R to a mark for the same candidate in T. In particular, while the permutations may move marks around, the composition of the two permutations is constrained as follows: no two elements belonging to the same ballot (in the same row in R) are mapped to two elements belonging to the same candidate (the same row in T). Let $\lfloor x \rfloor$ represent the greatest integer less than or equal to x (that is, the *floor* of x).

Definition 19 (Pointer-based back-end)**.** *A pointer based back-end consists of two random permutations $\pi_1, \pi_2 : \mathbb{Z}_{n\times c} \to \mathbb{Z}_{n\times c}$ such that*

$$\forall i, j \in \mathbb{Z}_{n\times c}, i \neq j \textit{ having } \lfloor \frac{i}{c} \rfloor = \lfloor \frac{j}{c} \rfloor \Rightarrow \lfloor \frac{\pi_2(\pi_1(i))}{n} \rfloor \neq \lfloor \frac{\pi_2(\pi_1(j))}{n} \rfloor \tag{5.1}$$

Eq. 5.1 from definition 19 says that Note that no group operation (such as modulo addition or permutation composition) is performed on the payload, since the payload is degenerated to a single bit simple representing a flagged/non-flagged state.

We now describe how the pointer based back-end implements the steps that it is responsible for in the protocol described in Chapter 4:

1 The back-end is set-up: the pointer-based back-end secretly computes π_1 and π_2 and publishes on the SPBB the commitments to them $(i, \mathfrak{C}(\pi_1(i)))$, $(i, \mathfrak{C}(\pi_2(i)))$. For each function committed to, the commitments are published in the ascending order of the index i.

2 The back-end gives any necessary information to the front-end: the pointer-based back-end computes $\pi_2 \circ \pi_1(i)$, $\forall i \in \mathbb{Z}_{n\times c}$ and gives the results to the front-end. Each result is of the form $(i, \pi_2 \circ \pi_1(i))$.

4 Both the back-end and the front-end are checked for correctness: an unpredictable subset $\mathbb{A}$ of indices $i \in \mathbb{Z}_n$ is generated in a public manner. For every $i \in \mathbb{A}$ the back-end publishes on the SPBB $(i, \pi_1(j), \pi_2(\pi_1(j)))$, where $j = i \times c + k, \forall k \in \mathbb{Z}_c$

5-10 The front end is used to vote; receipts are sent to the back-end

11 The back-end publishes cleartext ballots: for each vote i on a receipt, the back-end secretly computes $\pi_1(i)$ and $\pi_2 \circ \pi_1(i)$, sorts both outputs and publishes them on the SPBB.

12 The back-end proves that the published receipts are equivalent to the cleartext ballots: for all the post-images of π_1 (pre-images of π_2), an unpredictable publicly-verifiable choice is made to open the commitment to exactly one of i (the pre-image) or $\pi_2(\pi_1(i))$, (the post-image). The back-end then opens the commitment.

All the integrity and privacy results in Sect. 5.2 about the PunchScan back-end are valid for the pointer-based mixnet (Theorem 1, corollary 1, corollary 2 and Theorem 2 for integrity results and Theorem 3, Theorem 4, lemma 7, lemma 8 and Theorem 9 for privacy results). The proofs remaining essentially unchanged (with π_1 in place of D_1 and π_2 in place of D_2). Instead of ballots, however, the results apply to individual votes.

The design of the pointer based back-end is inspired by the contest partitioning idea presented in Sect. 5.2.9, but the idea is taken to an extreme: instead of separating simply the contests, each candidate is treated as a contest, is separated and follows a path that is independent of the paths taken by other candidates, even on the same ballot. In particular, because an under-vote travels by itself and does not carry with it any other vote-marking pattern, and because it can end up in any of the unmarked positions. we perform no special analysis for under-voted ballots or ballots voted in certain patterns for the pointer-based back-end, as its design prevents revelation of information in this form.

Most of the generalizations for improvement of the PunchScan back-end work equally well for the pointer-based back-end. Multiple pairs of the (π_1, π_2) permutation can be created, linked or unlinked, as presented in Sect. 5.2.5. The integrity and privacy consequences are the same. Recovery from failed audits is possible as presented in Sect. 5.2.8, and the choice of commitment schemes results in the same consequences presented in Sect. 5.2.10. Also, the techniques used to audit the back-end without any privacy loss presented in Sect. 5.2.7 are applicable.

5.3.2 Advantages and disadvantages

The pointer-based back-end is heavily inspired by the PunchScan back-end. Hence it inherits many of its properties in terms of the primitives used, efficiency and the central nature of the back-end. However, it also has some distinctive characteristics.

The pointer-based mixnet is much simpler from a mathematical point of view; instead of a function with two components—a permutation and an operation—it uses only a single component, the permutation. On the one hand this is easier to implement. Perhaps more importantly, it is much easier to explain to people who do not have a background in computer science or mathematics. The physical model presented in Chapter 11 can be easily understood by any middle-school pupil.

The only cryptographic primitive that the pointer-based back-end uses is a commitment function (no encryptions, no digital signatures). All the data is deterministically generated so as to allow for its reconstruction in subsequent steps. Step 1 of the pointer-based back-end is inherently slower than the corresponding step in the PunchScan back-end. On the other hand, step 11 is inherently faster; once the two permutations are recreated, no other computation needs to be performed.

The main advantage of the pointer-based back-end over the PunchScan back-end is the ability to naturally eliminate the Italian attack described in Sect. 5.2.9. For contests in which more than one candidate can be selected, even the intra-contest patterns are eliminated. However, the relation between voted candidates within one contest or between related contests can be kept explicitly if desired.

A new advantage offered by the pointer-based back-end is the ability to perform the initial steps without knowing the details of the election. The back-end can be set-up and audited without knowing how many races or how many candidates are in each race. We call this property "lazy ballots style". This may be important in very large elections or where the ballot styles are different for each jurisdiction.

As with the PunchScan back-end, the main disadvantage of the pointer-based back-end is its centralized nature. However, as with PunchScan, it is possible to distribute the authority via a threshold secret sharing technique.

5.4 Performance analysis

We have implemented the three back-ends presented in this chapter and we have measured how performance varies when parameters are changed. For a public citizenry that demands a rapid election turn around, the time required to process votes in the voting booth as well as after the completion of the election is critical. Hence, in this section, we focus on the most practical performance analysis measure: we choose a typical medium-sized election, and measure the time it takes to run our implementations on a personal computer for various total numbers of ballots. While some performance boost may be obtained by using a better implementation or faster machine, the orders of magnitude are expected to remain the same. All the tests were

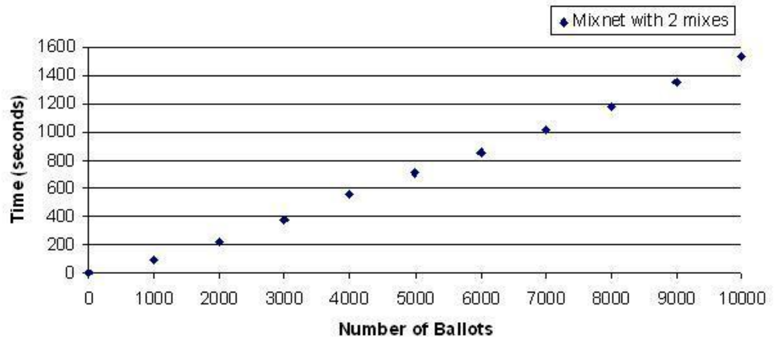


Figure 5.5: Onion mixnet back-end performance

conducted on a Dell Inspiron E1505 laptop, equipped with Intel Core Duo 1.73Ghz and 1Gb of RAM. All of the implementation is in Java, using the Eclipse framework. The Bouncy Castle open-source library was used to implement cryptographic functionality. The same election was used in all three cases: ten races having between 2 and 11 candidates per race, for a total of 38 candidates.

5.4.1 Performance analysis of onion mixnets

All the measurements were performed while keeping the following constant:

1. the image size, 600×400 (pixels)
2. the image color scheme, black and white
3. the font (Courier New, 18pts, Bold)
4. the number of mixes was fixed at 2

Fig. 5.5 presents the time taken to transform the encrypted votes into cleartext votes as a function of the number of ballots processed (note that this is the actual number of ballots cast in the election).

On average it takes around a tenth of a second per ballot to run through the two mixes. As expected, the time grows linearly with the number of ballots. Each mix input is represented by a PKCS#7 enveloped data message. This is necessary because the number of mixes can change; the size of the input increases with the number of mixes, and public key schemes do not typically directly accommodate variable size input. AES 192 in Cipher Block Chaining mode is used for the symmetric-key cipher, and RSA 1024 as the asymmetric-key cipher.

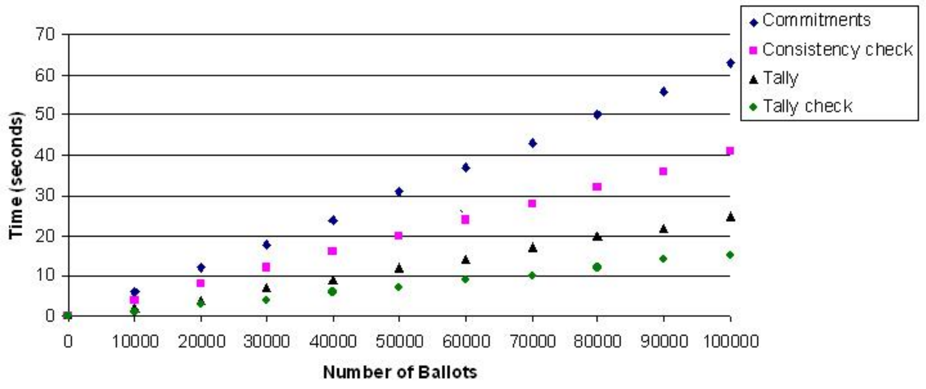


Figure 5.6: PunchScan back-end performance

While there are ways to improve the speed of an onion-mixnet back-end, it still remains inherently slow because of the asymmetric and symmetric descriptions that are required at each step, for each ballot. One way to improve the speed is to use the mixnet in pipeline mode; the input can be divided into smaller batches and the batches are injected one by one into the mixnet such that all the mixes work in parallel after the pipe gets full, as opposed to working strictly in serial in the original model.

5.4.2 Performance analysis of punchscanian mixnets

As described in Sect. 5.2 the PunchScan back-end can be implemented such that only four steps have to be performed (steps 1 and 2 can be combined). This was first described in [FCS06], and the four steps are referred to as "the four meetings". The back-end can be designed so that the entity that is responsible for the back-end may be a shared entity and may exist for four brief moments in time, when a number of trustees get together and collectively contribute to building the entity.

We have measured the time it takes for our implementation of the PunchScan back-end for each of the four meetings, as a function of number of ballots. The results are presented in Fig. 5.6. The number of ballots the election starts with is plotted on the X axis. Half of these ballots are spoiled when the consistency check is performed (step 4); we assume that the remaining half gets voted entirely. Thus the number of cast ballots is half the number of ballots plotted on the X axis; the reader should interpret Fig. 5.6 with this in mind.

A general practical concern is the time it takes to get the results of the election. Our experiments show that the PunchScan back-end is capable of decrypting 2000 ballots per second. The PunchScan back-end is faster by an order of 300 than the

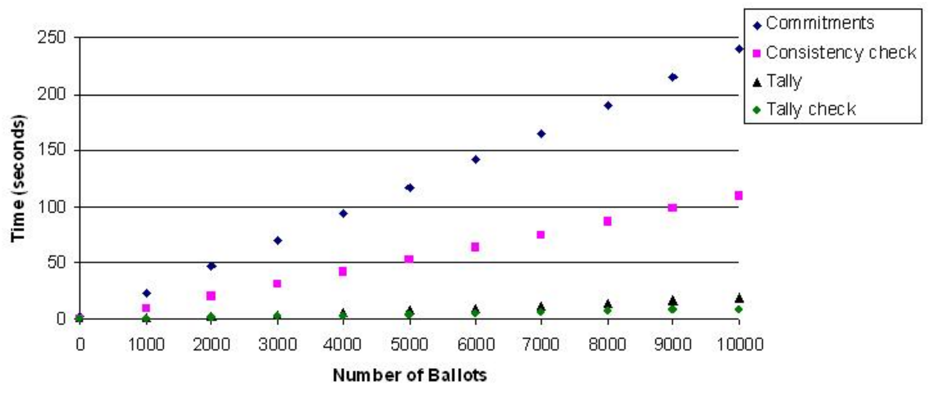


Figure 5.7: Pointer based back-end performance

onion mixnet back-end. For even the biggest counties, with four million voters, this means getting the results in at most 35 minutes.

Of all four steps performed by the PunchScan back-end, the step computing commitments, during the initial setup, is the most expensive (in terms of the time it takes to compute). This step is never required after the initial set-up, however. For example, when the consistency of the back-end is checked—(step 4 and step 12)—it is not required to reconstruct the commitments. The step can hence can take place long before the election, and its cost need not be a concern. Also note that the computation of the tally, the only step required in "real time" requires neither computation nor opening of commitments.

5.4.3 Performance analysis of pointer mixnets

Because the pointer-based mixnet is very similar to the PunchScan back-end, we have been able to simulate it using a particular configuration of an election. We have created a ballot with a single race and a single candidate in that race; if we group 38 such ballots and allow only 10 of each group to appear (because there are ten races with a single candidates per race), and use the PunchScan back-end to simulate such an election, for all performance purposes, we have simulated an election that uses a pointer-based back-end. Fig. 5.7 presents the results as a function of number of ballots. As with the PunchScan simulations, the X axis represents the number of ballots generated; only half of these ballots are used for voting.

Five thousand ballots are tallied in about twenty seconds, or two hundred and fifty ballots per second. This is about one order of magnitude slower than the PunchScan back-end, but about one and a half orders of magnitude faster than the onion mixnet.

Table 5.1: Properties of various back-ends

	Onion mixnets	**Punchscanian mixnet**	**Pointer mixnet**
Distributed authority	Yes	No	No
Paths	Dynamic	Static	Static
Back-end related to the ballot	No	Yes	Yes
Efficiency	Low	High	High
Lazy ballot style	Yes	No	Yes
Cryptography used	Symmetric and asymmetric encryption	Commitments	Commitments

As with PunchScan, commitments are only computed in the initial step; as can be seen from Fig. 5.7, the initial step takes the most time.

5.5 Conclusions

We have presented three back-ends that conform to the rules of our secure voting framework, as presented in Chapter 4. The onion mixnet has been widely studied in the past, hence our focus has been on the PunchScan and pointer-based back-ends. We have formally defined the PunchScan mixnet, proving its integrity and privacy properties. We have presented a new back-end—the pointer-based back-end—derived from the PunchScan back-end, that is simpler to explain and solves some of the potential privacy problems. We presented new techniques that allow for integrity verification while giving privacy guarantees, as well as techniques that allow for a tradeoff between integrity, privacy and performance. We have described new general techniques for protecting privacy, including how to generally avoid the Italian attack, and how to perform a randomized partial audit so as to prevent information leakage. Finally we have performed a performance analysis of our implementation of each of the three back-ends.

Table 5.1 summarizes the advantages and disadvantages of the three types of mixnet-based back-ends presented in this chapter.

Chapter 6

Front-ends

The front-end is the entity responsible for ballot production and voter interaction. In the general protocol presented in Chapter 4, about half of the steps are performed by the front-end. The front-end ensures that each voter gets a correctly formed ballot, and provides the voter with a receipt that she can later check, without leaking any information about how any voter voted.

The ballots produced by the front-end in step 5 of the general protocol in Chapter 4 fall in two categories. The first category is that of symmetric ballots: those that have two (or more) parts, each part with the same amount of information, and each part sufficient to recover the vote. On visual examination, when the two parts are 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 (or more) parts, where the amount of information is unequally distributed among the parts, and a specific part is needed to recover the vote. In this category too, the voter visually examines both parts of the ballot laid out in a particular manner (e.g. side by side) to confirm that the encrypted vote represents her ballot. We define symmetric and asymmetric ballots more formally below.

Definition 20 (Symmetric ballot). *A symmetric ballot is a ballot that has n parts and:*

- *when combining the n parts the cleartext 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*
- *any single part can become the receipt*

Definition 21 (Asymmetric ballot). *An asymmetric ballot is a ballot that has n parts and:*

- *when combining the n parts the cleartext 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*
- *a designated part always becomes the receipt*

We describe two techniques that can be used in step 7 to ensure that the ballots were printed correctly in step 5. The first technique is a cut-and-choose protocol on the set of available ballots at the polling place and handed out to voters. This technique works for both symmetric and asymmetric ballots, and is very simple: anyone who wishes to chooses, at random, a sample of ballots from all the ballots. These chosen ballots are clearly marked as audited or spoiled ballots and the front-end is required to open all commitments made during the set-up (step 3) that are related to the chosen ballots. If the commitments are not opened correctly, the audited ballots are irrefutable proof of error.

There are two main issues with the cut-and-choose technique. First, it can be used to mount a denial of service attack: so many auditors ask to check printing correctness that no more ballots remain available for casting votes. Second, a single auditor may not be able to, in practice, perform a printing audit at all the polling places. Both issues can be mitigated by having sufficient ballots on hand and allowing each auditor to randomly choose a limited, but statistically significant, number of ballots, at a randomly-chosen subset of the polling places.

Theorem 16. *[Misprinting ballots] Let n be the number of ballots available for voters in a polling place, out of which k are printed incorrectly. If r ballots are chosen uniformly at random from the n ballots and checked for printing correctness, the probability that all r are correctly printed is at most $min[(1-\frac{r}{n})^k, (1-\frac{k}{n})^r]$*

Proof. Follows directly from lemma 20. □

The second technique applies only to symmetric ballots. Because the ballot is formed from many parts, all containing the same amount of information, and any representing the receipt, a cut-and-choose protocol can be applied to a single ballot: each voter chooses, at random, one part of the ballot as the receipt and also performs on it the printing correctness check. The following two theorems assume that a symmetric ballot is used and that the printing correctness audit is performed by voters checking their receipts for printing correctness.

Theorem 17 (Misprinting symmetric ballots). *Assuming a symmetric ballot with p parts, let k be the number of ballots misprinted out of the total of n cast ballots. If r voters check their receipt, then the probability of not detecting any of the misprinted ballots is*

$$\sum_{x=0}^{k}(1-\frac{1}{p})^{x}\left(\begin{array}{c} r \\ x \end{array}\right)\prod_{i=0}^{x-1}\frac{k-i}{n-i}\prod_{i=0}^{r-x-1}\frac{n-k-i}{n-x-i} \tag{6.1}$$

Proof. The proof can be found in chapter 10. □

For example if a ballot with two parts is used, 3000 ballots are cast, 150 ballots are misprinted, and 150 voters check their receipts for printing correctness, the probability that the records on the SPBB are consistent with the receipts (i.e. the front-end cheated on 150 ballots and gets away with it) is less than 2.14%. Note that the theorem refers to the number of misprinted ballots that were cast during the election. If there are misprinted ballots that were not used, then this would not be detected but it would not affect the election outcome either.

In step 8, there are two types of receipts the voter can get: proof receipts and indication receipts. We define them formally below.

Definition 22 (Proof Receipt). *A proof receipt is a receipt held by the voter that is non-repudiable by the voting system.*

One type of proof receipt is one that is produced and digitally signed by the voting system itself (or by an election official). When a voter holds a proof receipt that is inconsistent with the information on the public bulletin board, and the signature scheme used is assumed secure, it is sufficient evidence that the bulletin board contains erroneous information. This, in turn, is an irrefutable indication of an error in the computation of the tally.

Definition 23 (Indication Receipt). *An indication receipt is one which can be used to trigger additional investigation if there is a discrepancy between the proof receipt and the public data.*

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.

The proof receipt is transferable. The voter may choose to hand over the proof receipt that she got to an organization that she trusts, thus delegating the responsibility of checking that the receipt is correctly posted on the SPBB. This may be a great advantage in practical settings.

An indication receipt is non-transferable. Only the voter knows that the indication receipt was indeed formed correctly from the ballot she cast. (Because the receipt is not authenticated, the voter can simply make up a receipt.) An entity collecting indication receipts would have to trust that voters are providing correct receipts. When this entity checks and sees that the information on the receipt is not correctly posted on the SPBB, it would not know if the voter lied or if the information on the SPBB really is corrupted. The organization may choose to file a complaint in any case, but the cost of filing such complaints will typically be greater than with proof receipts because some of these complaints will be found to be false.

6.1 Attaching the PunchScan front-end to various back-ends

An informal model for the PunchScan front-end, the voting ceremony that the voter has to follow to cast a valid ballot and other informal aspects related to the PunchScan front-end were presented in Sect. 3.6.3.

We now describe a rigorous model for a single contest. We start with notation. We abuse the notation $\mathbb{Z}_n$ to denote the set of numbers from 0 to $n-1$, and not the class of residues modulo n. If x is a vector, we denote the p^{th} position in the vector by v_p. We denote by $\_$ an argument of a function that is not important for the current presentation; example: $\mathcal{F}(i,\_) = (i,\_)$ means that the function $\mathcal{F}$ does not change the first argument, regardless of what the second argument is. This notation does not imply anything about the $\_$ on the left of the equal sign and the $\_$ on the right (e.g. that can be different or equal).

As with the notation for back-ends, let V^* be the ordered set of candidates in a particular contest, and let the cardinality of the set be $|V^*| = c$, the number of candidates. The order of the candidates in V^* is the order in which they appear on the ballot. Let $V = \emptyset \bigcup V^*$, where $\emptyset$ is a vote for nobody (we consider that V^* does not have an explicit "none of the above" vote).

We assume that there are n "virtual ballots" that the front-end obtained from the back-end. Let $T : \mathbb{Z}_n \times V^* \rightarrow \mathbb{Z}_n \times \mathcal{X}$ be a bijective function, where $\mathcal{X}$ is a set of cardinality c whose elements can be graphically represented, and any two different functions $(T(i, \_), T(j, \_))$ are independent,$\forall i \neq j \in \mathbb{Z}_n$. Let $B : \mathbb{Z}_n \times \mathcal{X} \rightarrow \mathbb{Z}_n \times \mathbb{Z}_c$ also be a bijective function, where any two different functions $(B(i, \_), B(j, \_))$ are independent.

$T(i, \_)$ represents the association of candidates to symbols on the top pages of the ballot i and $B(i, \_)$ represents the order in which the symbols appear on the bottom page of ballot i.

Let k be the number of candidates that the voter is allowed to select from V^*. Let $V^k = V \times V \times ... \times V$, the k times cartesian product of V with itself. This represents the non-over-votes that the voter can choose. For particular contests, some conditions can be placed on the cartesian product; for example, for any non-cumulative voting, no candidate except Ø can appear more then once in any of the tuples of the cartesian product, i.e. for any $v^k \in V^k$, $v^k_p \neq v^k_q$, $\forall p \neq q$ indices in v^k.

Definition 24 (PunchScan front-end)**.** *Let $\mathcal{F} : \mathbb{Z}_n \times V^k \rightarrow \mathbb{Z}_n \times \mathbb{Z}_c^k$ be a function such that:*

1. $\mathcal{F}(i, \_) = (i, \_)$ $\forall i \in \mathbb{Z}_n$
2. $\mathcal{F}(i, v)_{p+1} = 0$ *if and only if* $v_p = Ø$.
3. $\mathcal{F}(i, v)_{p+1} = B \circ T(i, v_p)$ *if and only if* $v_p \in V^*$

Then $\mathcal{F}$ is called the front-end of the PunchScan system.

We now describe how the PunchScan front-end implements the steps that it is responsible for in the protocol described in Chapter 4:

3 The front-end is set-up: based on information obtained from the back-end in step 2, the PunchScan front-end secretly computes $T(i, \_), B(i, \_)$, for all $i \in \mathbb{Z}_n$ and publishes on the SPBB the commitments to them $(i, ``T'', \mathfrak{C}(T(i, \_)))$, $(i, ``B'', \mathfrak{C}(B(i, \_)))$. For each function committed to, the commitments are published in the ascending order of the index $i \in \mathbb{Z}_n$.

4 Both the back-end and the front-end are checked for correctness: this step is only necessary if the functions T and B in the previous step were generated by the front-end based on the information it got from the back-end. If so, an unpredictable subset $\mathbb{A}$ of indices $i \in \mathbb{Z}_n$ is generated in a public manner. For every $i \in \mathbb{A}$ the front-end publishes on the SPBB $(i, T(i, \_), B(i, \_))$.

5 The front-end prints the ballots and distributes them to the polling places: for each ballot i, the front-end prints $T(i, \_)$ on the top page and $B(i, \_)$ on the bottom page. The pages are constructed as described in Sect. 3.6.3

6 Ballots are assembled: each top page is paired with the bottom page that has the same serial number.

7 Ballots are handed out to the voters, voted, and the front-end proves that the ballots were printed correctly:

   (a) The voter is allowed to choose any ballot i from the available ballots

   (b) Before seeing a ballot, a voter commits to which page T or B she wants to keep as a receipt. Assume the voter chooses the top page.

   (c) The front-end publishes the information from that page on the SPBB. Because the voter chose the top page for serial number i, the front-end publishes $T(i, \_)$.

   (d) The voter receives the ballot and goes into the privacy of a voting both.

   (e) The voter chooses her cleartext vote v and computes $(i, w) = B \circ T(i, v)$ (marking the holes that have the symbols corresponding to her favorite candidates).

8 Ballots are disassembled and a receipt is produced for the voter: the voter separates the top page from the bottom page of the ballot, the page that the voter chose in the previous step is scanned and kept by the voter as a receipt, while the other page is securely destroyed (e.g. shredded).

The PunchScan front-end uses a symmetric two-part ballot and produces a proof receipt. Thus the general integrity theorems for these types of ballots and receipts apply (e.g. Theorem 17). The following privacy results also hold.

Lemma 18. *Let w be a random element in V^*. For any $(i, T(i, \_), \mathcal{F}(i, v))$, there exists $B(i, \_)$ such that $\mathcal{F}(i, v) = B \circ T(i, w)$*

Proof. The proof follows directly from lemma 5. □

We assume that the page that the voter destroys is always kept secret; the following aspects are of paramount importance for privacy reasons:

1. There is no leakage of information about $T(i, \_)$ or $B(i, \_)$ in the chain of custody from the printing facility producing ballots (which knows both $T(i, \_)$ and $B(i, \_)$) to the voters.

2. No poll worker ever sees the $B(i, \_)$ if the voter chooses the top receipt, and $T(i, \_)$ if the voter chooses the bottom

3. $B(i, \_)$ is securely destroyed if the voter chooses the top receipt, and $T(i, \_)$ if the voter chooses the bottom, such that recovery is not possible.

The poll worker has to supervise the destruction of the page that was not chosen as a receipt, to ensure that the voter does not destroy a dummy page and secretly keep the original, which would allow the reconstruction of the cleartext vote v.

Theorem 19 (PunchScan ballot secrecy)**.** *Having the assumptions above, the receipt created by the PunchScan front-end does not reveal anything more than the number of candidates the voter chose.*

Proof. The proof can be found in Chapter 10. □

6.1.1 PunchScan front-end with PunchScan back-end

The PunchScan front-end coupled with the PunchScan back-end is the combination that was originally described in [PH06]. The mathematical relation that links the front-end with the back-end is:

Definition 25 (PunchScan voting system)**.** *Let $\mathcal{D}$ be a PunchScan back-end as per definition 16 and $\mathcal{F}$ a PunchScan front-end as per definition 24. We say the two make up a voting system if and only if $\forall i \in \mathbb{Z}_n, \exists j \in \mathbb{Z}_n$ such that $\mathcal{D} \circ \mathcal{F}(i, v) = (j, v)$ $\forall v \in V^k$.*

In other words, if the output of the front-end is input to the back-end, the back-end's output is a shuffle of the front-end input.

We now describe the steps in the general protocol that are involved in this linking:

1. The back-end gives any necessary information to the front-end: the PunchScan back-end gives $(i, \mathcal{D} = D_2 \circ D_1(i, \_))$ to the front-end

2. The front-end is set-up: the PunchScan front-end computes $T(i, \_)$ and $B(i, \_)$ such that $\mathcal{D} \circ B \circ T(i, v) = (j, v)$.

6.1.2 PunchScan front-end with pointer-based back-end

In step 2 of the general protocol, the PunchScan front-end gets from the pointer-based back-end a permutation $\pi : \mathbb{Z}_{n\times c} \rightarrow \mathbb{Z}_{n\times c}$. In step 3 the PunchScan front-end is setup as follows:

1. The input to the permutation π is divided into n consecutive sets, each set having c elements. Each set represents a ballot with c candidates.
2. For each of these sets, called ballots from now on, the values $i\%c$ are associated with the values $\lfloor \pi(i)/c \rfloor$—the floor of the quotient of the output of the permutation when divided by the number of candidates—is computed (see Fig. 5.4). This results in a new permutation for each ballot $\pi' : \mathbb{Z}_c \rightarrow \mathbb{Z}_c$
3. The PunchScan front-end computes the inverse of this permutation π'^{-1} and generates T and B such that $B \circ T = \pi'^{-1}$.

6.1.3 PunchScan front-end with onion mixnet back-end

In step 2 of the general protocol , the PunchScan front-end gets from the onion mixnet the public keys of each mix. In step 3 the PunchScan front-end is setup as follows:

1. for each $i \in \mathbb{Z}_n$ it randomly generates $T(i, \_)$ and $B(i, \_)$
2. for each pair ($T(i, \_)$,$B(i, \_)$) the PunchScan front-end computes $M(i, \_) = (B \circ T)^{-1}$.
3. for each $M(i, \_)$ the PunchScan front-end computes M_j such that $M(\_, \_) = M_1 \circ M_2 \circ ... \circ M_k$, where k is the number of public keys received in the previous step and the order of the mixes in the mixnet is $1, 2, ..., k$. Note that M_j does not have the first index (the initial serial number of the ballots). This is because the mixnet itself is going to determine the path that is taken through the mixnet.
4. For each ballot, the front-end computes the onion associated with it:$\mathcal{D}_j = \{M_j; \mathcal{D}_{j-1}\}_{K_j}$, where K_j is the public key of the j^{th} mix and $\{X\}_K$ represents the encryption of X with the key K.
5. The PunchScan front-end publishes all the onions computed in the previous step, $\mathcal{D}_k$, along with all the commitments for $T(i, \_)$ and $B(i, \_)$.

When the front-end is checked for correctness in step 4 of the general protocol, it reveals $T(i, \_)$,$B(i, \_)$ along with all M_js for that particular ballot. Then the onion can be recreated and compared to the one published.

6.1.4 Generalization

The authority that generates the PunchScan front-end is centralized. Even if a secret sharing technique can be used to recreate the authority only for brief moments in time, for those brief moments there still is one entity that knows all the information. Additionally, the entity that is printing the ballots knows both T and B for each ballot.

In this section we explore the possibility of having a truly distributed authority for the PunchScan front-end. We find that, while this is possible in theory, in practice, creating a usable ballot for the voter is difficult as the number of distinct authorities increases. This work was first described in [IPSC07]

The main idea is simple: use an authority to generate T and a separate authority to generate B. For a clear distinction, there is a separate back-end for each authority. Each back-end is able to partially transform the receipt of the voter; the final result, after all the back-ends perform transformations, is a cleartext ballot.

Intuitively, the front-end authority that creates a transformation is coupled with a back-end that inverts this transformation. The final ballot is the result of a series of such transformations. The classical PunchScan front-end consists of two transformations T and B (two pages, top and bottom). Assuming that the voter kept as a receipt the top page T, and that the receipt is public so that the voter can check it, the front-end would apply the (now public) transformation T to the receipt, and passes it to the back-end associated with the opposite page, B, to transform the result into a cleartext ballot.

An important observation is that the front-end that generated the transformation that the voter kept as a receipt will always apply its transformation first, while the other back-end will transform the result subsequently. Because the page that the voter chooses as a receipt is unpredictable, it follows that the operations performed by the front-ends should commute, so that the order in which the transformations are performed is not important. In particular, permutation composition is not commutative, but addition modulo c is. Thus a general permutation of candidate ordering will not be valid as an encryption of the vote (for T or B), as permutation composition is not, in general, a commutative operation. On the other hand, however, the composition of cyclic permutations is commutative, hence T and B could correspond to cyclic permutations.

Note that the authority that produces the clear ballot still retains the power to link a receipt to a cleartext vote. To solve this problem, we need to use a larger number of authorities. In general, n front-end and back-end authorities can be used, one front-end corresponding to a particular back-end, with the condition that the operations performed are commutative. The ballots would then consist of n transformations (pages) and the voter would choose one as a receipt, destroying all the other $n-1$ pages (transformations). The cleartext can be recovered from the receipt by the $n-1$ back-ends.

This generalization allows the voter to simply choose any pages she wants to form a ballot. While in the classical PunchScan, the two pages would have to bear the same serial number, with this generalization, any combination of serial numbers makes sense, as long as the pages were generated by different front-ends. This greatly simplifies the need for a strict chain of custody to protect privacy from the printing facilities to the polling places.

The practicality of having a ballot formed from n parts is debatable. Even with $n = 2$, it has been argued that the ballot is impractical to use. The generalization proposed in this section has theoretical value and we do realize that its practicality is debatable.

6.2 A new front-end: Attaching codes to candidates

One characteristic of the PunchScan front-end is that the scanner that reads the receipt does not see the cleartext vote. From a privacy point of view, this is an advantage, since any uniquely identifying information that the voter may leave on that page cannot be linked to a cleartext vote. From a reliability point of view this is a disadvantage, since the scanner cannot compute a tally and report it at the end of the day.

In this section, we suggest a new front-end that has these two properties reversed: the scanner reads a cleartext ballot while the voter keeps an encrypted receipt. This is easily achievable by modifying the PunchScan front-end slightly. Instead of having two functions T and B, only the T function is used. Recall that, for every ballot i the T functions associates a set of symbols to the candidates on the ballot. To

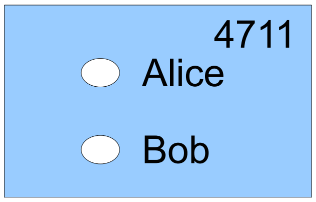


Figure 6.1: A typical optical scan ballot. Candidates are listed in a fixed order across all ballots and there is a designated location next to each candidate that a voter marks.

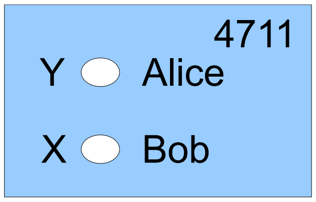


Figure 6.2: Scantegrity Ballot: on a regular optical scan ballot, next to each oval there is a different symbol.

simplify the presentation, we assume that the set of symbols consists of the letters of the English alphabet.

We assume that the order of the candidates on the ballot is fixed and the same on all ballots. Any standard optical scan ballot format is of this kind (see Fig. 6.1).

Definition 26 (Scantegrity front-end)**.** *Let $T : \mathbb{Z}_n \times V^* \rightarrow \mathbb{Z}_n \times \mathcal{X}$ be a bijective function, where $\mathcal{X}$ is a set whose elements can be graphically represented, and any two different functions $(T(i, \_), T(i', \_))$ are independent,$\forall i \neq i' \in \mathbb{Z}_n$. We call function T a Scantegrity front-end.*

For ballot i, $T(i, j)$ corresponds to candidate j (see Fig. 6.2). Note that knowledge of $T(i, j)$ does not reveal any information about $T(i', j)$ for $i \neq i'$; that is, the code for candidate j on ballot i does not reveal any information about the code for candidate j on ballot i' (see Fig. 6.3). Because $(T(i, \_)$ and $T(i', \_))$ are independent, the knowledge that a vote was cast for any particular symbol provides no statistical advantage in determining which candidate got the vote.

After the voter marks a vote for the desired candidate, the voter may write down the symbols next to the candidate. For all the voted candidates on each ballot, the front-end will compute $T(i, \_)$ and publish the symbols that are next to the voted candidates, which should be the same symbols that the candidate wrote down.

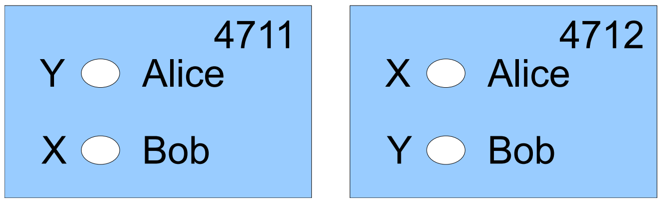


Figure 6.3: **Different Scantegrity Ballots.** On two different ballots, the order of the symbols associated with the candidates may be different.

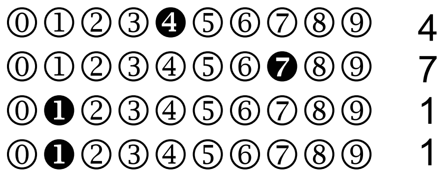


Figure 6.4: **Representing the serial number on an optical scan ballot.** This format allows a mark sense scanner to read it by checking which digits are black.

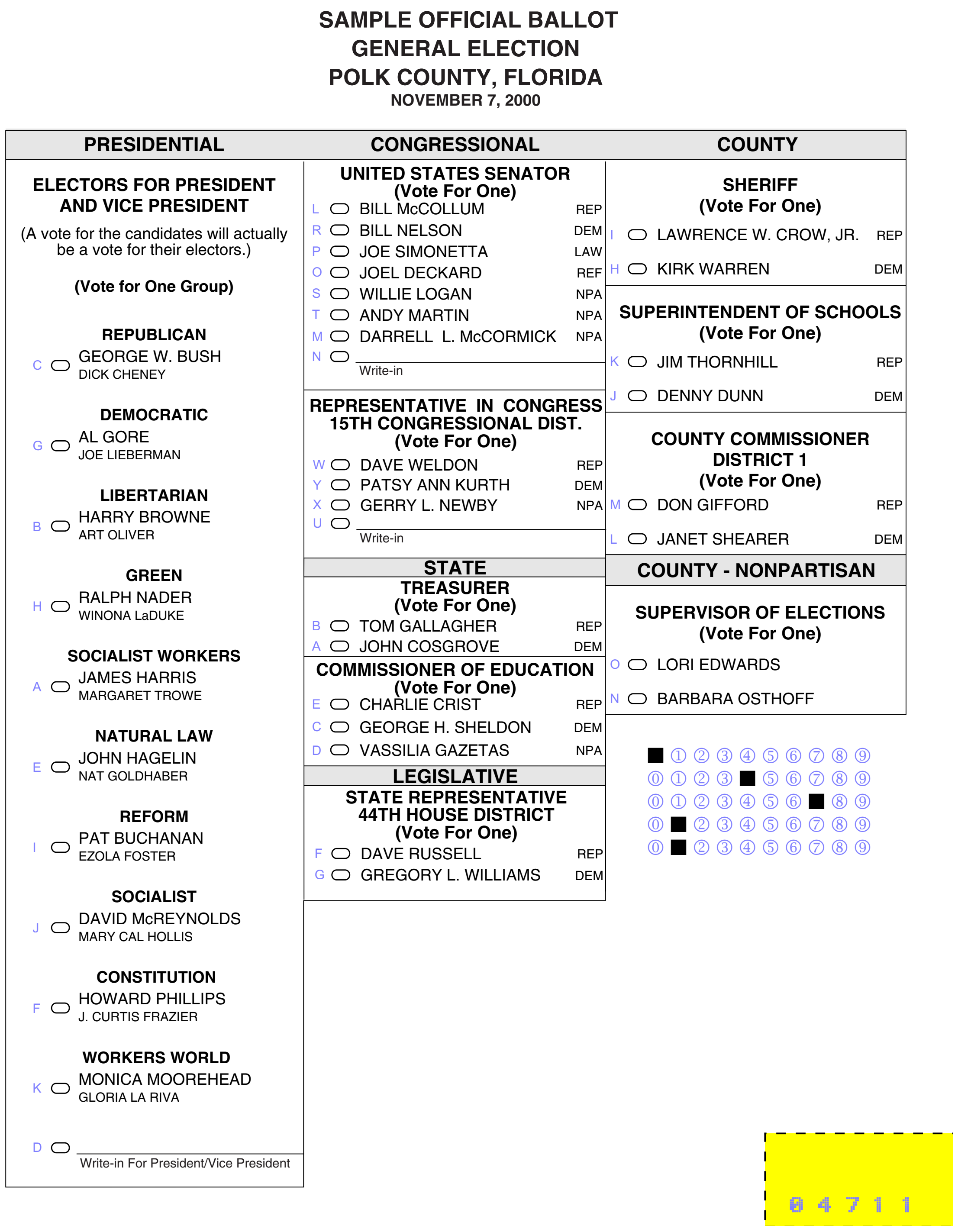
The Scantegrity ballot is an example of an asymmetric ballot. Thus the security techniques and properties of an asymmetric ballot, a described in Chapter 6 apply.

The serial number of each ballot is printed in a manner that allows for easy scanning using Marksense technology, commonly used for optical-scan voting systems. This is illustrated in Fig. 6.4. Marksense technology attempts to determine if a geometric shape is filled in or not. Sets of shapes can be defined such that only one shape is allowed to be filled in for each contest on a first-past-the-post ballot. The output of the scanner is an *electronic ballot images (EBI)* which is a list of each predefined shape at a given geometrical position, and its state—filled or unfilled. For our method, the information stored by the precinct scanners will be the positions that were filled in, where some of these positions represent the serial number.

A ballot adapted from a real election is shown in Fig. 6.5.

6.2.1 Dispute resolution

If the scanner reading and interpreting the optical scan ballot is equipped with a printer, and can read the symbols directly from the ballot, it can print the symbols next to the marked candidates using the on-board printer. This would effectively produce a proof receipt. The voter would need to check that the symbols on the receipt are the symbols next to the candidates that she voted for. We note that



SAMPLE OFFICIAL BALLOT
GENERAL ELECTION
POLK COUNTY, FLORIDA
NOVEMBER 7, 2000

PRESIDENTIAL

ELECTORS FOR PRESIDENT AND VICE PRESIDENT

(A vote for the candidates will actually be a vote for their electors.)

(Vote for One Group)

REPUBLICAN
C ○ GEORGE W. BUSH
DICK CHENEY

DEMOCRATIC
G ○ AL GORE
JOE LIEBERMAN

LIBERTARIAN
B ○ HARRY BROWNE
ART OLIVER

GREEN
H ○ RALPH NADER
WINONA LaDUKE

SOCIALIST WORKERS
A ○ JAMES HARRIS
MARGARET TROWE

NATURAL LAW
E ○ JOHN HAGELIN
NAT GOLDHABER

REFORM
I ○ PAT BUCHANAN
EZOLA FOSTER

SOCIALIST
J ○ DAVID McREYNOLDS
MARY CAL HOLLIS

CONSTITUTION
F ○ HOWARD PHILLIPS
J. CURTIS FRAZIER

WORKERS WORLD
K ○ MONICA MOOREHEAD
GLORIA LA RIVA

D ○ ______
Write-in For President/Vice President

CONGRESSIONAL

UNITED STATES SENATOR
(Vote For One)

L ○	BILL McCOLLUM	REP
R ○	BILL NELSON	DEM
P ○	JOE SIMONETTA	LAW
O ○	JOEL DECKARD	REF
S ○	WILLIE LOGAN	NPA
T ○	ANDY MARTIN	NPA
M ○	DARRELL L. McCORMICK	NPA
N ○	______ Write-in	

REPRESENTATIVE IN CONGRESS
15TH CONGRESSIONAL DIST.
(Vote For One)

W ○	DAVE WELDON	REP
Y ○	PATSY ANN KURTH	DEM
X ○	GERRY L. NEWBY	NPA
U ○	______ Write-in	

STATE

TREASURER
(Vote For One)

B ○	TOM GALLAGHER	REP
A ○	JOHN COSGROVE	DEM

COMMISSIONER OF EDUCATION
(Vote For One)

E ○	CHARLIE CRIST	REP
C ○	GEORGE H. SHELDON	DEM
D ○	VASSILIA GAZETAS	NPA

LEGISLATIVE

STATE REPRESENTATIVE
44TH HOUSE DISTRICT
(Vote For One)

F ○	DAVE RUSSELL	REP
G ○	GREGORY L. WILLIAMS	DEM

COUNTY

SHERIFF
(Vote For One)

I ○	LAWRENCE W. CROW, JR.	REP
H ○	KIRK WARREN	DEM

SUPERINTENDENT OF SCHOOLS
(Vote For One)

K ○	JIM THORNHILL	REP
J ○	DENNY DUNN	DEM

COUNTY COMMISSIONER
DISTRICT 1
(Vote For One)

M ○	DON GIFFORD	REP
L ○	JANET SHEARER	DEM

COUNTY - NONPARTISAN

SUPERVISOR OF ELECTIONS
(Vote For One)

O ○	LORI EDWARDS
N ○	BARBARA OSTHOFF

■ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
⓪ ① ② ③ ■ ⑤ ⑥ ⑦ ⑧ ⑨
⓪ ① ② ③ ④ ⑤ ⑥ ■ ⑧ ⑨
⓪ ■ ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
⓪ ■ ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

0 4 7 1 1

Figure 6.5: **Ballot adapted from Florida's 2000 Polk County Election[pol07].** Attached to every ballot is a chit with the ballot serial number.

regular optical scanners are usually equipped with a printer to be able to produce a printout of the tally at the end of the day.

We now consider the case when the scanner is not equipped with a printer, or if there is no scanner at all at the polling place (perhaps there is one at the central headquarters). In such cases, the voter would have to produce the receipt herself. Each page that the ballot is printed on has a portion specially designed to assist the voter in producing a receipt. We call this portion "the chit"[1]. Using a regular pen, the voter can write-down on the chit the symbols next to the candidates she voted for. The chit has preprinted on it the serial number of the ballot and special printing that can assist the voter: for example it can have for each race on the ballot a placeholder where the voter can write-down the symbols.

Before the ballot is placed into the optical scanner (or simply dropped in a ballot box), the voter, in the presence of an election official, detaches the chit from the rest of the ballot. This can be done by using a pattern scissors instead of a straight one. The election official would then rubber stamp the chit, to certify that it was indeed detached from the cast ballot. The voter keeps the chit and uses it to later check that the symbols she wrote down are correctly posted on the SPBB.

If the voter sees the correct symbols on the SPBB, she can be sure that:

1. The scanner properly read her ballot
2. The front-end correctly transformed her cleartext vote into the symbol she sees
3. The front-end correctly handed out her receipt to the back-end, i.e. the chain of custody from the scanner to the SPBB was unbroken.

If the voter does not see the symbols she wrote down, she can file a complaint. We now present a mechanism that allows the voter to irrefutably prove that her receipt was not correctly posted, or for the front-end to prove that the voter wrote down a wrong symbol. The protocol starts by having the voter file a complaint containing the serial number of her ballots. A period of time must elapse, until many (all) complaints are gathered.

For all the ballots (identified by serial numbers) that have the same symbols posted on the SPBB for the contested race (e.g. the posted symbol is X, but the voter claims it should be different, e.g. Y), the front-end retrieves the paper ballots from the storage. Each ballot is placed into an envelope that hides the part of the

[1]The PunchScan team won the “Best election technology component” prize at VoComp 2007 for this tear-off chit

ballot where the names of the candidates and the voter's marks are, but it exposes the portion where the chit was detached from, along with the serial number of the ballot. Each voter provides a chit that is checked to have a valid rubber stamp on it and each chit is matched with the paper ballot provided by the front-end. The patterned cut is checked to match and, if further investigation is needed, a fiber and chemical analysis is performed. If a valid stamped chit does not match the ballot with the same serial number, this is irrefutable proof that the front-end substituted the paper ballot.

Once it is determined that the ballots in the envelopes match all the chits, each ballot is transferred into a second envelope that hides the serial number of the ballot and the portion where the chit was separated from, but shows the marks the voter made, along with the symbols next to the chosen candidates. The ballots are shuffled before entering this second stage. The symbols revealed should be the same on all ballots, and should be the symbols that were posted on the SPBB. If so, the front-end just proved that all the voters complaining about the revealed ballots mis-wrote the symbols. If not all the symbols are the same, then it means that the voter's complaint is valid and an error occurred (either the scanner misread the ballot, the chain of custody was broken, etc).

This technique allows for the front-end to make a single proof for all the ballots that have the same symbols posted on the SPBB and voters complained about. However, it requires that the voter keep the physical chit and hand it over physically to the person that files the complaint (this person need not be the voter, but simply someone trusted by the voter) and that person be present when the front-end performs the physical proof.

6.2.1.1 Revealing the codes only for the voted candidates

When looking carefully at the tear-off chit, we notice that only the symbols next to the candidates that the voter voted for are on it. Thus the voter only needs to know these codes and not the codes next to the candidates the voter did not cast a vote for. Inspired by this observation we present a variation of the Scantegrity ballot, Scantegrity II, that simplifies the dispute resolution protocol presented in Sect. 6.2.1.

We print the symbols *inside* the oval using invisible ink. When the oval next to the candidate is marked with a special pen, the code appears. More precisely, the background of the oval turns black to allow the optical scanner to detect it, while the code inside the oval turns yellow. The voter can only read the code if she fills in the oval, and thus the scanner will detect the filled in oval as well. We call this front-end Scantegrity II, where II stands not only for the second version but also for Invisible Ink.

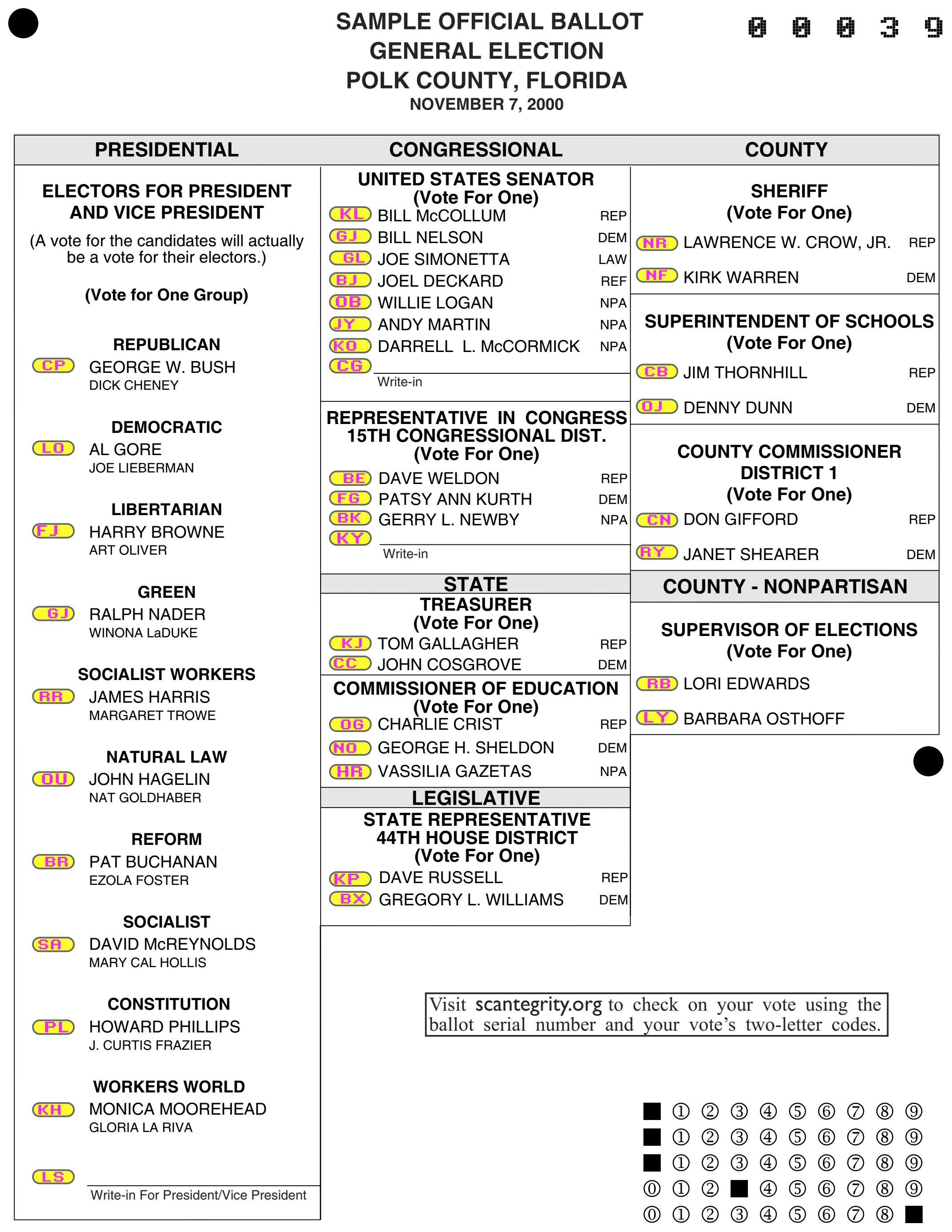
An unmarked ballot is presented in Fig. 6.7(a), and a marked one in Fig. 6.7(b). Fig. 6.6 has a sample ballot adapted from the Polk County, Florida 2000 election. This is the electronic version of the ballot that will be sent to the printer (hence it shows all codes).

The same result can be achieved with scratch-off surfaces: the oval is printed with a black background and a yellow code in it and this is covered with a white scratch-off surface. To mark, the voter removes the scratch off surface next to her favorite candidate, much as she would reveal the numbers on a lottery ticket.

The dispute resolution protocol can be modified to take advantage of this technique: while the first version of Scantegrity used something the voter had (the chit), Scantegrity II can use something the voter knows, the codes next to the candidates she voted for. If the codes appear to the public as randomly generated and thus are hard to guess on any ballot, then the fact that the voter knows a code that was printed next to a candidate on a certain ballot means that the voter marked that oval.

If the voter checks the online SPBB and notices that the code she saw next to the candidate she voted for is not correctly posted on the SPBB, she can fill out an online petition, containing the serial number of the ballot along with the code the voter claims she saw. The front-end will then open all the commitments corresponding to that serial number. If all the commitments check and if the code provided by the voter is not among the revealed codes, it means that the voter made a mistake while writing the code down (or that the voter invented a complaint). If the code provided by the voter is among the revealed codes, it means that either the voter simply happened to guess the code or the front-end cheated and was caught. Depending on how many complaints are registered for the same ballot, and how many valid complaints are registered in total, a statistical trigger can determine if the voter simply guessed the code or not. For example if a single complaint is registered for a ballot and the code turns out to be valid, and the number of complained ballots is small, it is very likely that the front-end cheated. A brief description of such statistical triggers can be found in [CCC$^+$08].

An advantage of using invisible ink is that the ink can be made such that it reacts slowly in time with the chemicals in the marking pen. For example, the codes in the oval can themselves turn totally black 15 minutes after the oval was marked. Thus,



SAMPLE OFFICIAL BALLOT
GENERAL ELECTION
POLK COUNTY, FLORIDA
NOVEMBER 7, 2000

0 0 0 3 9

PRESIDENTIAL

ELECTORS FOR PRESIDENT AND VICE PRESIDENT

(A vote for the candidates will actually be a vote for their electors.)

(Vote for One Group)

	REPUBLICAN
CP	GEORGE W. BUSH DICK CHENEY
	DEMOCRATIC
LO	AL GORE JOE LIEBERMAN
	LIBERTARIAN
FJ	HARRY BROWNE ART OLIVER
	GREEN
GJ	RALPH NADER WINONA LaDUKE
	SOCIALIST WORKERS
RR	JAMES HARRIS MARGARET TROWE
	NATURAL LAW
OU	JOHN HAGELIN NAT GOLDHABER
	REFORM
BR	PAT BUCHANAN EZOLA FOSTER
	SOCIALIST
SA	DAVID McREYNOLDS MARY CAL HOLLIS
	CONSTITUTION
PL	HOWARD PHILLIPS J. CURTIS FRAZIER
	WORKERS WORLD
KH	MONICA MOOREHEAD GLORIA LA RIVA
LS	Write-in For President/Vice President

CONGRESSIONAL

UNITED STATES SENATOR
(Vote For One)

KL	BILL McCOLLUM	REP
GJ	BILL NELSON	DEM
GL	JOE SIMONETTA	LAW
BJ	JOEL DECKARD	REF
OB	WILLIE LOGAN	NPA
JY	ANDY MARTIN	NPA
KO	DARRELL L. McCORMICK	NPA
CG	Write-in	

REPRESENTATIVE IN CONGRESS
15TH CONGRESSIONAL DIST.
(Vote For One)

BE	DAVE WELDON	REP
FG	PATSY ANN KURTH	DEM
BK	GERRY L. NEWBY	NPA
KY	Write-in	

STATE

TREASURER
(Vote For One)

KJ	TOM GALLAGHER	REP
CC	JOHN COSGROVE	DEM

COMMISSIONER OF EDUCATION
(Vote For One)

OG	CHARLIE CRIST	REP
NO	GEORGE H. SHELDON	DEM
HR	VASSILIA GAZETAS	NPA

LEGISLATIVE

STATE REPRESENTATIVE
44TH HOUSE DISTRICT
(Vote For One)

KP	DAVE RUSSELL	REP
BX	GREGORY L. WILLIAMS	DEM

COUNTY

SHERIFF
(Vote For One)

NR	LAWRENCE W. CROW, JR.	REP
NF	KIRK WARREN	DEM

SUPERINTENDENT OF SCHOOLS
(Vote For One)

CB	JIM THORNHILL	REP
OJ	DENNY DUNN	DEM

COUNTY COMMISSIONER
DISTRICT 1
(Vote For One)

CN	DON GIFFORD	REP
RY	JANET SHEARER	DEM

COUNTY - NONPARTISAN

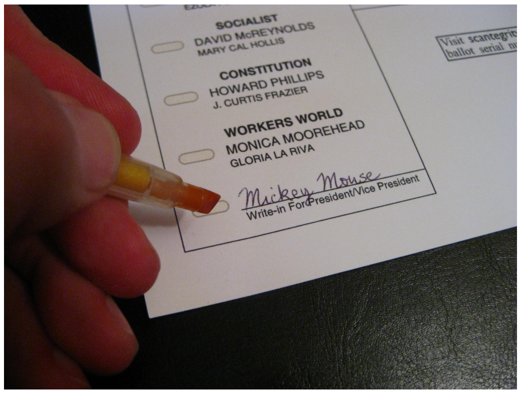
SUPERVISOR OF ELECTIONS
(Vote For One)

RB	LORI EDWARDS
LY	BARBARA OSTHOFF

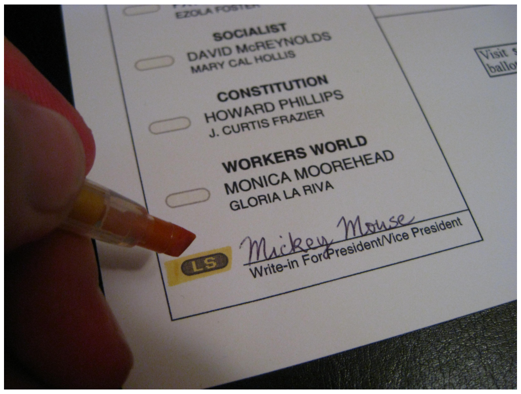
Visit scantegrity.org to check on your vote using the ballot serial number and your vote's two-letter codes.

■ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
■ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
■ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
⓪ ① ② ■ ④ ⑤ ⑥ ⑦ ⑧ ⑨
⓪ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ■

Figure 6.6: **Scantegrity II ballot adapted from Florida's 2000 Polk County Election[pol07].** The symbols are printed inside the oval using invisible ink.



(a) An unmarked Scantegrity II ballot.



(b) A marked Scantegrity II ballot.

Figure 6.7: Marked and unmarked Scantegrity II ballot. Photos courtesy of Aleks Essex

at the end of the day, when the paper ballots are taken out of the ballot box, all the ballots look exactly like optical scan ballots and no symbols are visible on them.

6.2.2 Advantages and disadvantages

The Scantegrity system inherits all the properties of the traditional optical scan systems, but also adds new ones. Just as some of the properties of optical scan voting systems are beneficial and some are not, the newly introduced properties also fall into two categories.

We begin by listing the desirable properties of traditional optical scan systems and the beneficial additions provided by Scantegrity II. First, the voting interface is highly familiar to the voters. In the 2006 US election, more than 56% of counties used traditional optical scan, accounting for almost 49% of the cast ballots. While the exact details of the system vary (some require voters to fill in the bubble, others to connect the ends of two arrows, etc), Scantegrity II can be added to all these variations in an unobtrusive way. The equipment used to scan and register the cast ballots would be compatible with the new way of marking the ballots, making the overall system very cheap in itself and making the addition of the Scantegrity or Scantegrity II layer negligible in terms of cost, when compared to the base optical scan. While Marksense technology may be sufficient, a full graphic scanner capable of reading the serial number and the confirmation codes written with the Latin alphabet is the logical next step.

Some of the Marksense scanners currently in use do not store the cleartext Electronic Ballot Images (EBIs) - the records that show exactly where dark portions have been detected on each ballot. Rather, these scanners tabulate the ballots as they are scanned and they only report the totals at the end of the day. Scantegrity needs the raw EBIs to be able to recover the codes that the voters saw while marking their ballots in the voting booth; hence such scanners are not compatible with Scantegrity and would need a software upgrade and possibly a hardware upgrade to add the memory needed to store these records. Scanners that keep individual records for each ballot can also report an independent tally when voting is completed.

A fixed order of candidates on all ballots is allowed, but variable orders are not precluded. A manual recount can be performed in case such an audit is ordered. In the first Scantegrity system proposed [CEC$^+$08], a strict chain of custody is needed to protect privacy, both before the ballots reach the voters and after the ballots are cast. In Scantegrity II, there is no need for a strict chain of custody after voting (except for an eventual accurate manual recount), since all the ballots are indistinguishable by the human eye after they are taken out of the ballot box at the end of the voting period (if slow reacting ink is used). This is a clear advantage over the PunchScan or Prêt à Voter front-ends.

Any kind of contest that is supported by a classic optical scan is also supported by Scantegrity, but Scantegrity does not offer the same level of integrity to all types of contests. In particular, for write-in candidates the voters can only check that a write-in vote was cast, but not if it was correctly recorded or counted.

The verifying process that the voters may follow to check that their ballots were recorded as cast is very easy: they type in the serial number of the ballot and check that the confirmation codes they wrote on the chits are the same as the ones on the public bulletin board. If a discrepancy is observed, the complaint process does not rely on the voter to bring any physical evidence; instead, the voter simply fills-in a form on-line and submits the codes that she thinks are correct. While a single voter that notices a discrepancy may not be sufficient to trigger an investigation, it is possible to set up a statistical trigger, based on the number of complaints received.

The main drawback of the Scantegrity systems is that they require voter effort to create the receipt. Voting is one action; creating a receipt to take home and check later is a distinct second action that the voter has to perform if she wants to check that her voter was recorded as cast. On the other hand, the receipt creation and verification is opt-in: the voter needs to take the extra step only if she wishes to be an active part of the integrity check.

6.2.3 Misprinting in Scantegrity

In all the situations below, misprinted ballots are detected via the general printing audit for asymmetric ballots described in Chapter 6. Also, as with optical scan voting systems, we assume that voters notice if the order of candidates on the ballot is not the canonical order. The analysis below is to demonstrate that additional methods exist to detect some forms of cheating. It assumes that opscan and Scantegrity are separate and the cheaters from one are not the same with the cheaters from the other.

1. The order of the candidates is correct, the order of the codes is incorrect and the scanner detects black ovals: The dark oval will be translated by the back-end into a code that is different from the one the voter saw. Thus the voter will detect that cheating occurred.

2. The order of the candidates is correct, the order of the codes is incorrect and the scanner reads codes but not filled ovals: cheating is not detected by the voter. The voter sees on the bulletin board the code that the scanner read and there is only one tally, the one computed by the back-end (since the scanner only reads the codes and is not able to compute an independent tally). Cheating of this kind is detected, however, through the printing audits.

3. The order of the candidates is correct, the order of the codes is incorrect and the scanner reads both codes and ovals: Assume that the tally is computed by the scanner based on the reading of black ovals and that the back-end gets from the scanner the codes, not the black ovals. Then the tally produced by the scanner is inconsistent with the tally produced by the back-end. Assume that the tally is computed by the scanner based on the reading of black ovals and that the back-end gets from the scanner the black ovals, not the codes. The dark oval will be translated by the back-end into a code that is different from the one the voter saw. Thus the voter will detect that cheating occurred. Also, Scantegrity itself will detect that an inconsistency occurred, since the ovals from the scanner do not translated into the codes the scanner provided. Note that the tally of the scanner will be identical to the tally of the back-end (both wrong).

4. The order of the candidates is incorrect, the order of the codes is correct and the scanner detects black ovals: cheating is not detected by the voter. The dark ovals are translated into correct codes that the voter sees on the bulletin board. The vote is interpreted differently than the voter saw, by both the scanner and

the back-end. Cheating of this kind is detected, however, through the printing audits.

5. The order of the candidates is incorrect, the order of the codes is correct and the scanner reads codes but not filled ovals: cheating is not detected by the voter. The voter sees on the bulletin board the code that the scanner read and there is only one tally, the one computed by the back-end (since the scanner only reads the codes and is not able to compute an independent tally). Cheating of this kind is detected, however, through the printing audits.

6. The order of the candidates is incorrect, the order of the codes is correct and the scanner reads both codes and ovals: cheating is not detected by the voter. It does not matter if the codes on the bulletin board come directly from the scanner or the scanner provided the black ovals and they were translated into codes, the codes the voter sees on the bulletin board are consistent with the codes the voter saw in the booth.

7. The codes are correctly assigned to candidates, but the order of the candidates is incorrect and the scanner detects black ovals. The dark oval will be translated by the back-end into a code that is different from the one the voter saw. Thus the voter will detect that cheating occurred.

8. The codes are correctly assigned to candidates, but the order of the candidates is incorrect and the scanner reads codes but not filled ovals: no cheating occurs. The codes the scanner reads are correctly transformed by the back-end into the votes the voter intended.

9. The codes are correctly assigned to candidates, but the order of the candidates is incorrect and the scanner reads both codes and ovals: Assume that the tally is computed by the scanner based on the reading of black ovals and that the back-end gets from the scanner the codes, not the black ovals. Then the tally produced by the scanner is inconsistent with that produced by the back-end. Assume that the tally is computed by the scanner based on the reading of black ovals and that the back-end gets from the scanner the black ovals, not the codes. The dark oval will be translated by the back-end into a code that is different from the one the voter saw. Thus the voter will detect that cheating occurred. Also, Scantegrity itself will detect that an inconsistency occurred, since the ovals from the scanner do not translated into the codes the scanner provided.

Note that the tally of the scanner will be identical to the tally of the back-end (both wrong).

6.2.4 Attaching the Scantegrity front-end to the onion mixnet back-end

In step 2 of the general protocol on page 68, the Scantegrity front-end gets from each mix of the mixnet its public key. In step 3 the Scantegrity front-end is setup as follows:

1. for each $i \in \mathbb{Z}_n$ it randomly generates $T(i, \_)$ and $M(i, \_) = T^{-1}$.
2. for each $M(i, \_)$ the Scantegrity front-end computes M_j such that $M(\_, \_) = M_1 \circ M_2 \circ ... \circ M_k$, where k is the number of public keys received in the previous step and the order of the mixes in the mixnet is $1, 2, ..., k$. Note that M_j does not have the first index (the initial serial number of the ballots). This is because the mixnet itself is going to determine the path that is taken through the mixnet.
3. For each ballot, the front-end computes the onion associated with it:$\mathcal{D}_j = \{M_j; \mathcal{D}_{j-1}\}_{K_j}$, where K_j is the public key of the j^{th} mix and $\{X\}_K$ represents the encryption of X with the key K.
4. The Scantegrity front-end publishes all the onions computed in the previous step, $\mathcal{D}_k$, along with all the commitments for $T(i, \_)$.

When the front-end is checked for correctness in step 4 of the general protocol, it reveals $T(i, \_)$ along with all M_js for that particular ballot. Then the onion can be recreated and compared to the one published.

6.2.5 Attaching the Scantegrity front-end to the punchscanian mixnet back-end

This combination has originally been proposed for Scantegrity and has been described in [PCE+08].

In step 2 of the general protocol on page 68, the Scantegrity front-end gets from the PunchScan back-end a set of functions that translate pairs of ballot serial numbers and votes (expressed as numbers) to shuffled cleartext votes. In step 3 the Scantegrity front-end is setup as follows:

1. The back-end gives any necessary information to the front-end: the PunchScan back-end gives to the front-end a function $\mathcal{D} : \mathbb{Z}_n \times \mathbb{Z}_c^k \to \mathbb{Z}_n \times V^k$, where $\mathcal{D} = D_2 \circ D_1(i, \_)$.
2. The front-end is set-up: the Scantegrity front-end generates $T : \mathbb{Z}_n \times V^k \to \mathbb{Z}_n \times \mathcal{X}$ such that $\mathcal{D} \circ T(i, v) = (j, v)$. The front-end publishes commitment to all $T(i, \_)$.

The function T is printed on the ballot. When the value of $T(i, v)$ is known (the voter's receipt), it is made public and transmitted to the back-end.

6.2.6 Attaching the Scantegrity front-end to the pointer-based back-end

This combination has been described in [CEC+08] and [CCC+08].

In step 2 of the general protocol on page 68, the Scantegrity front-end gets from the pointer-based back-end a permutation that translates numbers that are consecutively assigned to ballots to numbers that are consecutively assigned to candidates.

In step 3 the Scantegrity front-end is setup as follows:

1. The back-end gives any necessary information to the front-end: the pointer-based back-end gives a permutation $\pi : \mathbb{Z}_{n\times c} \to \mathbb{Z}_{n\times c}$, to the front-end.
2. The front-end is set-up: the Scantegrity front-end generates $T : \mathbb{Z}_n \times V^k \to \mathbb{Z}_n \times \mathcal{X}$ such that $\pi \circ T(i, v) = j$ and j represents a vote for v. The front-end publishes commitments to all $T(i, \_)$.

The function T will be printed on the ballot. When the output of the function T is known (the receipt the voter gets), it is made public and transmitted to the back-end.

6.3 Attaching the Prêt à Voter front-end to various back-ends

An informal model for the Prêt à Voter front-end, the voting ceremony that the voter has to follow to cast a valid ballot and other informal aspects related to the Prêt à Voter front-end were presented in Sect. 3.6.2.

The Prêt à Voter front-end can be modeled as a permutation of candidates, a function that associates c candidates to numbers from 0 to $c-1$, $P : V \rightarrow \mathbb{Z}_c$, where V is the set of candidates (clear votes). The inverse of this function is sorted by its input and the output is the order in which the candidates appear on the ballot.

We now describe how the Prêt à Voter front-end implements the steps that it is responsible for in the protocol described in Chapter 4:

3 The front-end is set-up: based on the information that it got from back-end in step 2, the Prêt à Voter front-end secretly computes $P_i : V \rightarrow \mathbb{Z}_c$, for all $i \in \mathbb{Z}_n$ and publishes on the SPBB the commitments to them $(i, \mathfrak{C}(P_i))$. For each function committed to, the commitments are published in the ascending order of the index $i \in \mathbb{Z}_n$.

4 Both the back-end and the front-end are checked for correctness: this step is only necessary if P in the previous step was generated by the front-end based on the information it got from the back-end. If so, an unpredictable subset $\mathbb{A}$ of indices $i \in \mathbb{Z}_n$ is generated in a public manner. For every $i \in \mathbb{A}$ the front-end publishes on the SPBB i, and P_i.

5 The front-end prints the ballots and distributes them to the polling places: for each ballot i, the front-end prints P_i^{-1} on the left side of the ballot.

6 Ballots are assembled: this step is not necessary as ballots are formed of a single page.

7 Ballots are handed out to the voters, voted and the front-end proves that the ballots were printed correctly:

(a) The voter is allowed to choose any number of ballots, choose one to vote and the rest to check the printing.

(b) The front-end publishes all the information about the audited ballots.

(c) The voter receives the ballot and goes into the privacy of a voting both.

(d) The voter choose her cleartext vote v and computes $P(v)$ (makes a mark next to the candidate on the right side of the ballot).

8 Ballots are disassembled and a receipt is produced for the voter: the voter separates the right side containing her marks from the left side containing the order of the candidates, keeps the right side and securely destroys the left side.

The Prêt à Voter ballot is an example of an asymmetric ballot with proof receipt. Thus the theorems and proof from Chapter 6 apply. The Prêt à Voter voting system was originally described [CRS05] in conjunction with an onion mixnet. In the following sections we describe how to couple it with the punchscanian and the pointer-based mixnets.

6.3.1 Punchscanian mixnet

In step 2 of the general protocol , the Prêt à Voter front-end gets from the PunchScan back-end a permutation $\mathcal{D}$. In step 3 the Prêt à Voter front-end is setup as follows:

1. The back-end gives any necessary information to the front-end: the PunchScan back-end gives $(i, \mathcal{D} = D_2 \circ D_1(i, \_))$ to the front-end

2. The front-end is set-up: the Prêt à Voter front-end computes $P(i, \_)$ such that $\mathcal{D} \circ P(i, v) = (j, v)$. Thus P_i is the inverse of $\mathcal{D}(i, \_)$.

6.3.2 Pointer mixnet

In step 2 of the general protocol , the Prêt à Voter front-end gets from the pointer-based back-end a permutation $\pi : \mathbb{Z}_{n\times c} \to \mathbb{Z}_{n\times c}$. In step 3 the Prêt à Voter front-end is setup as follows:

1. The input to the permutation π is divided into n consecutive sets, each set having c elements. Each set represents a ballot with c candidates.

2. For each of these sets, called ballots from now on, the values $i\%c$ are associated with the values $\lfloor \pi(i)/c \rfloor$—the floor of the quotient of the output of the permutation when divided by the number of candidates—is computed (see Fig. 5.4). This results in a new permutation for each ballot $\pi' : \mathbb{Z}_c \to \mathbb{Z}_c$

3. The Prêt à Voter front-end computes the inverse of this permutation π'^{-1} and sets $P = \pi'^{-1}$.

6.4 Attaching the visual ballot based on visual cryptography

The front-end based on visual cryptography is based on the exclusive or of two images. The image must be able to accommodate long names, or names that the voter can introduce as write-in option. Thus the number of pixels in the image tends to be fairly large. An overview of the front-end based on visual cryptography can be found in Sect. 3.6.1, along with a description of how to couple the front-end based on visual cryptography to the onion mixnet back-end. Half of the pixels in each of the two images that are xored are pseudo-randomly generated from a seed. The seed has to be long enough such that it is not easy to simply guess; a reasonable seed size is 128, such that the total number of possibilities for the pseudo-random string is 2^{128}.

This may impose certain restrictions on the combination between the front-end based on visual cryptography and the punchscanian and pointer-based mixnets.

6.4.1 Attaching the front-end based on visual cryptography to the pointer-based mixnet

The pointer-based back-end can transform a fixed set of integers representing encoded ballots into another fixed set of integers representing cleartext ballots. The size of the input set has to be equal to the size of the output set for the transformation to be a bijection. For efficiency reasons, the size of the sets cannot be very large, as the number of commitments that need to be computed and published is double the size of the set.

The size of the set is direct proportional to the number of ballots. Even in a very large election, the number of ballots does not tent to be so large that it is not practical (2^{30} ballots is a manageable number). Combining this with the size requirement from the front-end based on visual cryptography of approximately 2^{128}, we get approximately $2^{30} \times 2^{128} = 2^{158}$ possibilities, which is too large.

We see that the generality and length requirement of the front-end is in conflict with the efficiency and limited option set offered by the back-end. In a practical system, we suggest to have a less general front-end based on visual cryptography and keep the efficiency of the back-end. One way to reduce the generality of the front end is to allow a vote for one of a small number of candidates; this corresponds to one of a small number of possible input images. The setup for such a system is as follows:

1. The input to the permutation π is divided into n consecutive sets, each set having c elements. Each set represents a ballot with c candidates.

2. For each of these sets, called ballots from now on, the values $i\%c$ are associated with the values $\lfloor \pi(i)/c \rfloor$—the floor of the quotient of the output of the permutation when divided by the number of candidates—is computed (see Fig. 5.4). This results in a new permutation for each ballot $\pi' : \mathbb{Z}_c \rightarrow \mathbb{Z}_c$

3. The front-end based on visual cryptography computes how the image would look like if the voter would have voted for each of the n candidates and publishes commitments that tie each image with a preimage of π'. When the voter casts a vote for a particular candidate, the corresponding commitment is opened.

6.4.2 Attaching the front-end based on visual cryptography to the punchscanian mixnet

The front-end based on visual cryptography can be combined with the punchscanian mixnet as described in the previous section, but with loss of generality.

Inspired by the technique presented in Sect. 6.1.4, we now present a way to combine the front-end based on visual cryptography with the punchscanian back-end without losing the ability to represent write-in votes. In order to do this, we marginally modify the back-end; we set up a punchscanian back-end with all the principles of the original one, but especially designed for the front-end based on visual cryptography.

In particular the functions D_1 and D_2 are defined on the set $\{0,1\}^{|W|}$ where $|W|$ represents the number of pseudo-random pixels in the image and the function that composes the two is bitwise exclusive or. As presented in Sect. 6.1.4, two back-ends need to be set-up, one for the top page and one for the bottom page. This is not done for the same reason as in Sect. 6.1.4 (i.e. to allow separate authorities to print the separate pages of the front-end), but instead to be able to extract from the page that the voter kept as a receipt, the bits that were not pseudo-randomly generated and that need to be xored with $D_1 \circ D_2$ to recover half of the image that the voter saw[2]. Because the xor operation is commutative, the voter is free to keep any of the pages as a receipt. The back-end corresponding to the complementary chosen page is selected to decrypt the receipt.

[2]We assume that the image has enough redundancy that the intent of the voter can be unambiguously recovered from only half of the image

6.5 Special cases of front-ends

One of the biggest advantages of separating the front-end from the back-end with specified interfaces for each is that, for a given back-end, a variety of front-ends can be used, even in the same election. For example one of these front-end can be accessible to voters with disabilities while another can be used in a remote setting. The next sections contain a detailed description of how such front-ends could be constructed.

6.5.1 Accessible front-ends

In the previous paradigm, where each voting system was viewed as a monolithic block, the way the voters interact with the system had to be mimicked for voters with disabilities. Thus there would be separate descriptions for Prêt à Voter, PunchScan, Scantegrity, etc. In the new paradigm, a single accessible front-end has to exist for each back-end. Voters with disabilities do not have to mimic the actions performed on the front-end by other voters. All the votes will be processed by the same back-end; its output will contain cleartext ballots that are indistinguishable, in the sense that one cannot say if a particular ballot was cast by a voter with disabilities or not.

Inspired by the Benaloh challenge [Ben08], we present a simple front-end dedicated to voters who are visually impaired that can be easily adapted to work with all of the three back-ends presented in Chapter 5. We describe the voter experience, from the voter's point of view. We then analyze the scheme in term of cost and present its strengths and weaknesses.

A blind voter comes in and identifies herself. She is given a number (the poll worker tells the number to the voter). After having the number, the voter decides if she wants to have that ballot cast or spoiled. Assume that the voter wants to have the ballot cast. She goes into a private booth, where she uses her own cell phone to dial in a 1-800 number and types in the code she got from the poll worker. She hears: "to vote for Alice press 1, to vote for Bob press 2". She presses 2. She hears: "you voted for Bob". After she chooses all her favorite candidates she is asks to review her selections and to confirm her ballot. At this point the voter hears: "please be prepared to retain your confirmation code. The confirmation code for voting for Bob is K7W". Then the systems asked the voter: "Do you want to cast this ballot or to audit it?". The voter says she wants to cast it and she is done. Knowledge of the confirmation code is enough to check later that her vote was recorded as cast.

In the case of the PunchScan and Pointer based back-ends, there can be one confirmation code per candidate. In the case of an onion mixnet, there can be one confirmation code for the entire cast ballot. Before the election, the front-end commits to all the possible confirmation codes and to how each of the confirmation codes are going to be translated as inputs to the back-ends (numbers). Once a confirmation code is given to a voter, the corresponding commitments can be opened, and the input to the back-end is revealed.

If the voter wanted to audit the ballot, she would have to be prepared. After registering and saying that she would like to audit the given ballot (identified by the code the poll worker gives her), she would put her phone on speaker mode and ask a trusted poll worker to listen to her interaction with the system. Because the system at the other end of the line does not know if the voter is the only one listening, or if the phone is on speaker (essentially it does not know if the ballot is going to be audited or not), it has to give the voter the correct confirmation code for the chosen candidate. The cost of this type of accessible front-end is small. Open source tools allow the set up of a phone service, and a speaker phone is not very expensive.

The advantages of the scheme are as follows: the voter can use her own phone, one she is already accustomed to; it is extremely easy to explain to the voters and to the poll workers; it has low cost and is extremely easy to follow by a voter.

The weaknesses of this type of accessible front-end are: the poll worker that assists the voter in spoiling a ballot needs to be trusted - he cannot lie about what he hears (both what the voter or the machine is saying). A small set of people can distinguish between votes cast by blind voters and the rest of the cast votes, but to the general public these votes will be indistinguishable, in both their encrypted and cleartext forms. Another weak point is that the machine can spoil the ballot even if instructed to cast it and vice versa. Finally, the voter cannot be allowed to use an electronic phone, as phone logs would reveal how she voted.

As of now, the problem of a truly accessible voting system, with the properties of the paper-based ones described in this dissertation, is open.

6.5.2 Mail-in ballots

Mail-in ballots are becoming increasingly common around the world, in constituencies large and small. The motivation for this includes allowing those unable to visit the polling station (such as those in the military stationed overseas; expatriates; those less able) to cast votes from a remote location and to widen participation. The normal procedure for mail-in (or postal) ballots is that the materials (including ballot forms and a number of envelopes) are sent to registered voters, who have requested it, in the post. The voter is instructed to fill out the ballot form in private and to place it in an envelope. This envelope is then put in another envelope on which the identity of the voter is printed: the voter signs this outer envelope. This multi-layered document is subsequently sent, in yet another envelope, to the election authority. At the election authority headquarters, a number of clerks open the outermost envelopes of the received ballots. This reveals the identity of the voter who has posted the ballot and when the clerks have checked that she is eligible, the signed envelope is opened and the contents dropped into a ballot box. When all ballots have been dropped into the ballot box, it is shaken so that it is infeasible for anyone to guess the identities of voters. The contents of this ballot box can now be tallied together with ballots collected in polling stations.

The most obvious flaw in this process is the fact that the voter is not offered the secrecy of a voting booth. Also, as with traditional paper-based voting schemes, the voter must also place trust in people and procedures. The mail-in ballot procedure described earlier may arguably be a best-case scenario. Many places offer, in our view, even less secrecy and security. We describe a technique that works for Prêt à Voter, PunchScan and Scantegrity front-ends to provide the integrity guarantees of polling place voting, in a remote setting. We do not address the privacy issue of remote voting.

In the case of PunchScan the immediate technique to handle mail-in ballots is to allow the voter to mark her two-page form in the comfort of her own home using the required dauber. The voter then separates the two pages, chooses one and posts this to the election authority. This page is scanned but kept secret by the authority. After the close of the election a meeting of the election officials generates not the page that the voter has posted to the authority but its twin. In short, the page published on the web bulletin board will be the same as the one the voter has still got. It seems that this will provide the voter with a way to verify the inclusion of her vote in the tally, just as with those votes collected in polling stations.

However, in polling stations the page that the voter is going to keep is scanned and immediately published onto the web bulletin board. In the mail-in ballot case the other page is scanned and the first is generated from this. This means that if the voter deliberately marks the two pages separately, the receipt will not match the page published on the web bulletin board. As there is no way of detecting whether this is because the ballot form has been maliciously altered by the authority or the voter has marked the pages differently, the page kept by the voter cannot be regarded as a useful receipt. Furthermore, there are issues with the chain of custody of the page that has been posted in by the voter. When the authority receives the page a clerk will take it from its envelope and scan it. The clerk, who has seen this page, will be able to tell from the other page, published on the web bulletin board, how the voter has voted. Similarly, using the page she has kept, the voter will be able to prove, to a coercer, how she voted.

It was proposed in 2005 [CRS05] that Prêt à Voter can handle mail-in ballots simply by allowing the voter to fill out the normal ballot form in her home, tear along the perforation and then post the right-hand part to the election authority. This has the advantage over traditional mail-in ballots that the vote is encrypted when it reaches the election authority and the receiving clerk thus has no way of discerning the voter's choice(s). However, this does not offer the voter a receipt. Furthermore, it weakens coercion resistance even further as the destruction of the left-hand side of the ballot form, the randomized candidate list, cannot be enforced. Those voters who are able and willing to safeguard the secrecy of their vote will fill out the form in secret and, as soon as possible thereafter, destroy the candidate list; but some may be unable to do so. In order to provide the voter with a receipt that she can subsequently use to check the inclusion of her vote in the tally, this scheme may be extended with a carbon paper which is overlaid on the right-hand side of the ballot form. When the voter makes her marks on the ballot form these marks are thus made on two overlain sheets of papers. When she has made her marks and detached and destroyed the left-hand side of the form what remains are two copies of the encrypted receipts. The voter posts the original and keeps the carbon copy. It is quite clear that a malicious voter could mark the original and the carbon copy of the encrypted receipt separately and thus achieve the same attack as described in the above.

In Scantegrity, the mail-in version is the same as the polling place version. The voter marks the ballot normally and removes the chit that is attached to it, with the only difference that now the voter uses the two layer envelope (used for the Scantegrity back-end dispute resolution process) to send the ballot out. If the voter does not see the codes next to her candidates posted correctly[3], she has the chit and can provide it while filling a complaint, asking for her physical ballot to be retrieved, matched with the chit and inspected. Thus the Scantegrity front-end works unmodified in a remote setting.

Inspire by Scantegrity, we apply the same "chit" technique for PunchScan and

[3]We assume the voter sent in the ballot using certified mail, and she got a receipt when she handed out the envelopes to the poster worker.

Prêt à Voter. Thus, in essence, our technique is based on a chit, a corner of the ballot form, which holds the serial number of the form and is torn from it by the voter before the form is submitted. All the actions the voter must undertake in order to vote by mail-in ballot are thus as follows: The voter marks her choices on the ballot form and separates it into two, by detaching the pages or the columns from each other. In the case of Punchscan she randomly selects one page to keep as her receipt. She then detaches the chit from the receipt and puts this in a safe place, along with a note of the choice(s) she has made at each position on the form. The part of the form that must be destroyed is placed into an envelope that has a window through which the serial number of this part can be read.

Along with this envelope the encrypted receipt is placed in yet another envelope which bears the voter's identity. Having sealed this envelope the voter signs it to indicate to the election authority that she has cast her vote in accordance with the rules and so forth. This envelope is placed in an envelope which is pre-printed with the address of the receiving election authority and when this has been sealed it can be dropped in the nearest post box.

When the election authority receives the ballot, its outermost envelope is opened by a clerk who verifies the identity of the voter and her signature. If this check passes then the envelope is stripped off. The next check that must be performed is that the serial number of the part of the ballot form that has been placed in the innermost envelope matches the serial number of the encrypted receipt. This procedure is necessary to ensure that the voter returns the part of the ballot form that must be destroyed. In order to make this as secure as possible the election authority would have placed a rubber stamp onto the serial number of both parts of the form. If the serial numbers match, then the clerk shreds the envelope containing the discarded part of the ballot form and scans the encrypted receipt, causing it to be posted to the web bulletin board. The original encrypted receipt is then filed according to its serial number so that it may be retrieved with relative ease at some future point.

The voter is now able to visit the web bulletin board and call up her encrypted receipt, using the serial number printed on her chit. She can compare this to her notes. In the case where the voter finds that the online representation of her encrypted receipt does not match the notes she made at the time of voting, she may use her chit to verify that her receipt was scanned correctly. She, or her representative, can take the chit to the election authority. Under the scrutiny of independent—or bipartisan—auditors and media coverage, the election authority can retrieve the encrypted receipt from the archive and show, perhaps using a microscope or any level of forensic equipment required by the auditors, that the chit and encrypted receipt were once one piece of paper. When it has been shown that the chit matches the encrypted receipt the auditors can check that the contents of the encrypted receipt do match the electronic representation published on the web bulletin board.

In Scantegrity, which uses plain-text ballot forms, this check is rather tricky. However, in the case of Prêt à Voter and Punchscan, where the receipt is encrypted, it is safe simply to show the receipt to the voter or any representative that she might have mandated.

6.6 Randomization attacks

In this section we examine the effect of forcing a voter to bring back a certain type of receipt. For example, in PunchScan or Prêt à Voter, a mark in a fixed position may mean a vote for Alice on one ballot, and a vote for Bob on another ballot. An attacker may coerce voters to put their mark at a certain position, effectively enforcing a random vote. The coercer may also ask for a blank receipt. We show that the effect of random votes on election outcome is very similar to that of abstentions. In particular, that the rank ordering of votes between two candidates is maintained in a randomization attack on a particular fraction of voters if and only if it is maintained in an abstention attack on the same fraction of voters. The mathematical demonstration is only concerned with the number of votes, and therefore is valid for binary contests, choose one out of m or choose up to m candidates. The discussion is not valid for rank voting since there are multiple ways of "counting the votes", deciding the candidate ranking, and determining the winner when it comes to rank voting. Forcing registered voters not to vote is possible outside of the technical description of the voting system in many practical ways, thus the systems that are vulnerable to randomization attacks do not typically introduce new vulnerabilities in an election.

Note that the randomization attack can change the outcome of an election, when compared to abstention, in two particular instances. First, when the winner is required to win a majority of the votes cast, and second, when the winner is required to obtain a certain fraction of the total number of valid voters (not votes; that is, an abstention is a negative vote).

Assume that in an honest election, the results would be:

Candidate 1 - n_1 votes

Candidate 2 - n_2 votes

...

Candidate c - n_c votes

Let $n = n_1 + n_2 + ... + n_c$ be the total number of votes. Note that n is the number of votes, not the number of voters. In the case of "choose up to m candidates", the total number votes is m times the total number of voters (assuming that all the voters do choose m candidates). Let $n_1 > n_2 > ... > n_c$; so that candidate one is the winning candidate.

6.6.1 Forcing uniform random votes and uniform random silence

Assume Mallory buys a fraction $0 \leq f \leq 1$ of the total votes and forces them to become random votes. Then, candidate k will:

- Lose $n_k \times f$ votes, because these votes will now be random instead of in favor of n_k
- Win some votes from all the other candidates. From candidate i, $n_i \times f \times \frac{1}{c}$ will now be in favor of candidates k, for a total of:

$$\begin{aligned} & n_1 \times f \times \frac{1}{c} + n_2 \times f \times \frac{1}{c} + \cdots + n_c \times f \times \frac{1}{c} \\ = & \frac{f}{c} \times (n_1 + n_2 + ... + n_c) \\ = & \frac{f}{c} \times n \text{ votes.} \end{aligned}$$

In general n'_k, the number of votes for candidate k after buying a fraction f of random votes, is $n'_k = n_k - n_k \times f + \frac{f}{c} \times n$

For Mallory's favorite candidate, k, to win it would have to have the highest number of votes after the coercion attack. So, $\forall j, 1 \leq j \leq n, j \neq k, n'_k > n'_j$,

$$n_k - n_k \times f + \frac{f}{c} \times n > n_j - n_j \times f + \frac{f}{c} \times n \quad \Rightarrow \quad (n_j - n_k) \times f > n_j - n_k$$

If k would have been the winner even without the attack,$(n_j < n_k)$, then k would still be the winner after the attack (the inequality becomes $f < 1$, which is true).

If $n_j > n_k$, thus k would not be the winner before the attack, the inequality is a contradiction, because it would require $f > 1$, thus k cannot become the winner after the attack. In both cases this style of attack would not change the election results.

We prove that choosing random voters and forcing them not to vote at all would not result in an attack that would change the order of the candidates, just as above. Moreover, we show that the mathematical inequalities end up being the same.

When buying uniform random silence candidate k will lose $n_k \times f$ votes, because these voters will be forced to stay at home.

Thus $\forall k, n'_k = n_k - n_k \times f$. For Mallory's favorite candidate, k, to win it would have to have the highest number of votes after the coercion attack. So, $\forall j, 1 \leq j \leq n, j \neq k, n'_k > n'_j$,

$$n_k - n_k \times f > n_j - n_j \times f \quad \Rightarrow \quad (n_j - n_k) \times f > n_j - n_k$$

This is the exact same inequality as above. The same discussion applies.

6.6.2 Forcing nonuniform random votes and nonuniform random silence

We now focus on an attack that would succeed: assume that Mallory determines which voters will not vote for Mallory's desired candidates. She decides to buy random votes only from these *non-supporting* voters. The result is that candidate k does not lose any votes from his loyal voters (since none are targeted for the attack), and wins some votes from each of his opponents, because he picks up some of the random votes.

Thus, $\forall j, 1 \leq j \leq n, j \neq k$

$$\begin{aligned} n'_j = & \quad n_j - n_j \times f \\ & + \; n_1 \times f \times \frac{1}{c} \\ & + \; n_2 \times f \times \frac{1}{c} + \ldots \\ & + \; n_{k-1} \times f \times \frac{1}{c} \\ & + \; n_{k+1} \times f \times \frac{1}{c} + \ldots \\ & + \; n_c \times f \times \frac{1}{c} \\ = & \quad n_j - n_j \times f + \frac{f}{c} \times (n - n_k) \end{aligned}$$

Each candidate j loses some votes from their own voters being forced to randomly vote, and gains some votes from other candidates' voters being forced to randomly vote, but does not gain any votes from *k's* voters as they are not targeted in the attack. So candidate k only gains from this attack, and will end up with n'_k votes.

$$n'_k = n_k + \frac{f}{c} \times (n - n_k)$$

Generally, for any two candidates different from k, their relative rank will be preserved by this attack. If i was before j without the attack, i would still be before j after the attack. Mathematically, $\forall i, j$ such that $i > j, i \neq k, j \neq k$, if $n_i < n_j$ then

$$\begin{aligned} n'_i < n'_j &\Leftrightarrow n_i - n_i \times f + \frac{f}{c} \times (n - n_k) > n_j - n_j \times f + \frac{f}{c} \times (n - n_k) \\ &\Leftrightarrow (n_i - n_j) > (n_i - n_j) \times f \\ &\Leftrightarrow 1 > f(TRUE) \end{aligned}$$

Therefore, for k to win the elections, it would have to have more votes than candidate 1. Thus,

$$\begin{aligned} n'_k > n'_1 &\Rightarrow n_k + \frac{f}{c} \times (n - n_k) > n_1 - n_1 \times f + \frac{f}{c} \times (n - n_k) \\ &\Rightarrow n_1 - n_k < n_1 \times f \end{aligned}$$

$$\Rightarrow 1 - \frac{n_k}{n_1} < f \tag{6.2}$$

This is the fraction of the voters that have to be coerced to vote randomly by Mallory such that her favorite candidate, k, wins the election.

Now, we assume that Mallory coerces a random fraction of voters known not to vote with k, to not vote at all.

All the candidates other than k lose a fraction of their votes: $\forall j, 1 \leq j \leq n, j \neq k, n'_j = n_j - n_j \times f$, while candidate k keep all his votes: $n'_k = n_k$

Just as in the previous cases, to have candidate k win, we need

$$\begin{aligned} n'_k > n'_1 &\Rightarrow n_k > n_1 - n_1 \times f \\ &\Rightarrow n_1 \times f > n_1 - n_k \\ &\Rightarrow f > 1 - \frac{n_k}{n_1} \end{aligned}$$

same as Eq. 6.2.

Therefore, it takes the same effort to make k win, regardless if voters are paid to vote randomly or not to vote at all.

Note: If the fraction f is not the same for each candidate, the result is similar: $f_j > 1 - \frac{n_k}{n_j}$ for every $j < k$

We have shown that the ability for receipt-based voting systems to support purchase of a verifiable random vote, does not make it easier for a coercer to influence an election. A random vote has a similar effect to someone buying abstentions from voters, which can be verified by active voter lists or other known methods.

6.7 Conclusions

We have formally described the PunchScan and Scantegrity front-ends and shown that they are particular cases of symmetric and asymmetric ballots that produce proof and indication receipts, respectively. We proved what general integrity and privacy properties symmetric and asymmetric ballots have, and showed that the probability of cheating and getting away with it drops exponentially with the number of ballots cheated on. For four types of front-ends, PunchScan, Scantegrity, Prêt à Voter and the front-end based on visual cryptography, we showed how they can be attached to the three types of back-ends described in Chapter 5: the onion, punchscanian and pointer-based mixnets. We have thus shown that what were previously viewed as monolithic voting systems are in fact particular combinations of two general concepts: front and back-ends. Table 6.1 summarizes the advantages and disadvantages of the four types of front-ends.

We also presented a general front-end that is accessible by visually impaired voters and a technique that allows the voters that use mail-in ballots to have the same level of assurance as those who vote at a polling place. Finally, we showed that even though some of the front-ends presented here are susceptible to randomization attacks, mathematically, they are no worse than any other voting system that publishes the the list of active voters.

Table 6.2 summarizes the desirable qualities of the front-end back-end combinations presented in this thesis.

	Visual cryptography	**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
Voting machine knows cleartext votes/can compute 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

Table 6.1: Evaluation of various types of ballots

	Visual cryptography	**PunchScan**	**Prêt à Voter**	**Scantegrity II**
Onion mixnet	fixed order of candidates, accommodates write-ins, truly distributed, does not require a strict chain of custody to protect the privacy, the voting machine can report the tally	easy to check by voters, fixed order of candidates, easy to implement in practice, truly distributed	easy to check by voters, easy to implement in practice, truly distributed	easy to check by voters, fixed order of candidates, easy to implement in practice, voters are familiar with the interface, allows a manual recount, truly distributed
Punchscanian mixnet	fixed order of candidates, accommodates write-ins, the voting machine can report the tally, processes the votes very fast	easy to check by voters, fixed order of candidates, easy to implement in practice, processes the votes very fast	easy to check by voters, easy to implement in practice, processes the votes very fast	easy to check by voters, fixed order of candidates, easy to implement in practice, voters are familiar with the interface, allows a manual recount, processes the votes very fast
Pointer based mixnet	fixed order of candidates, the voting machine can report the tally, easy to understand how the tally is verified	easy to check by voters, fixed order of candidates, easy to implement in practice, easy to understand how the tally is verified	easy to check by voters, easy to implement in practice, easy to understand how the tally is verified	easy to check by voters, fixed order of candidates, easy to implement in practice, voters are familiar with the interface, allows a manual recount, easy to understand how the tally is verified

Table 6.2: Qualities of various combinations of front and back -ends

Chapter 7

Case study

7.1 The GSAÉD election: the first binding election using an E2E system

PunchScan is the first voter-verifiable end-to-end voting system used in a binding election. It was used by the *Graduate Students' Association / Association des étudiant(e)s diplômé(e)s* (GSAÉD) of the University of Ottawa in March 2007 for its general elections. This section describes the election.

After the launch of PunchScan 1.0 in November 2006, the PunchScan team pursued the use of PunchScan in a student election. Student elections were attractive as test cases for the use of voter-verifiable voting systems for several reasons. They provided a large enough number of voters with multiple polling places. Further, we expected that students would be open to innovative ways of capturing voters' intentions, that they would be passionate about their elections and would hence care about voter verifiability, and that the rules and regulations for student elections would be more flexible than those for public elections. We had discussions with a few universities (Georgetown University, University of Iowa) and discovered that, as in the public election world, the technologies used in student elections were very diverse: from hand counted paper ballots, to Internet voting. Some of the voting systems were developed in house (Georgetown University), while other universities (Marymount University) used a web site service.

At the time of the election, two PunchScan team members, Aleks Essex and Jeremy Clark were graduate students at the University of Ottawa. They presented the PunchScan system to the student body of GSAÉD, which voted unanimously to adopt the PunchScan voting system for the 2007 general election. Among the reasons cited for their decision were the desires to speed up the tally process, to increase the integrity of election results, to provide a means to identify double-voting, to increase voter turnout and to play a leading role in voting system research.

In consultation with GSAÉD, we arrived at several requirements for this election: five polling stations located at the University of Ottawa campus, and a fully bilingual ballot in English and French. The contested offices, names of candidates, and French translations were provided by GSAÉD.

In PunchScan the election officials only keep an electronic record representing the scanned and interpreted receipt image. GSAÉD had employed paper-based voting systems and had procedures for the handling, counting, and eventual disposal of paper ballots. The student body was keen to retain the use of paper records, to protect against power outages, hard drive failures, and other electronic denial-of-service attacks. It was agreed that a small space in the lower right-hand corner of the receipt would be reserved for making a copy of the marked positions. After scanning the receipt, it was to be placed in a printer that would print the marked positions together with the serial number on this corner. Using a pair of scissors, the poll worker detached and retained the corner before returning the receipt to the voter.

This requirement evolved into a stand-alone authentication mechanism of the receipt. Instead of using a straight scissors, we used patterned scissors. Because of the uniqueness of the starting point and angle, every time they are used, the poll workers and the voters obtain a unique pattern. We used the piece of paper we retained as a means of authentication against a counterfeit receipt. This way, the election authority has an easy and intuitive way of checking if a receipt was home-brewed or not. This chit cut with a patterned scissors later won "Best election technology component" at VoComp 2007. To guard the voter against election officials creating different paper records after the ballot casting ends, we also used a barcode-based digital signature printed onto the ballot receipt at the time that it was cast. At the polling place, the ballot receipt is scanned and the Polling Place software detects and encodes the serial and mark positions into an XML file. The ballot receipt is then placed in a printer and colorized. O' shaped overlays are printed on top of the ink daubs confirming their detection, and 'X' shaped overlays are printed over unmarked positions. The marked positions are then digitally signed and printed on the bottom left corner of the ballot receipt in a scannable barcode format. This way the voter is sure that she has a valid receipt and the election authority is sure that the voter cannot add (or remove) marks from the receipt. The keys used for the digital signature are generated each tim the scanner/computer are booted up and and self signed; the public key is included in both the starting and the ending reports, which are signed by all the poll workers.

The key used to sign the receipt was randomly generated by each polling place each time the voting software was started. The certificate associated with it was printed on the start-up report, and the printout was hand-signed by at least two poll workers. This procedure eliminated the need for a public key infrastructure. When the voting software was shut down, a closing report was printed, containing the number of scanned ballots, the number of cast ballots and the number of spoiled ballots. Both starting and ending reports were also time stamped.

In past GSAÉD elections, voters were authenticated at the polling place using local paper copies of the voter list. Voters' university-issued graduate student cards were used as the primary credential to vote. Although this scheme allows the eventual identification of students who cast more than one ballot at different polling stations, it does not allow a means by which to purge the double votes from the tally. GSAÉD had experienced problems with double voting in past elections, and had employed solutions such as the use of walkie-talkie radios to communicate the student numbers of voters in a real-time fashion. However, the local copies of voter lists were still being updated manually, which is inefficient given a list of 4000 eligible voters and 5 polling stations. The PunchScan team offered and implemented an alternative solution: a centralized electronic pollbook. A database of eligible student numbers was created and maintained on the `PunchScan.org` server. The poll worker interface was through a standard web browser and Internet connection. Using the campus wireless network, the poll workers were able to log into a password protected, SSL secured web form and perform voter list queries and updates. Querying a voter's student number would return one of three possibilities: a message indicating that the student number was not found in the database, an option to mark that individual as having voted, or an alert that that student had voted already.

Although it was agreed to be unlikely to occur in this election environment, the PunchScan team decided to test a countermeasure to "chain voting": a practice where the adversary gives the voter a pre-voted ballot to cast, and asks her to bring back an empty, valid ballot from the polling booth. To increase the difficulty of someone being able to remove a ballot from the polling station without detection, the PunchScan team developed special clipboards that employ a *Medeco*[1] cylinder lock affixed to the clipboard to lock the top right corner of the ballot to the clipboard. Before the voter

[1]Medeco High Security Locks sponsored the PunchScan project with several locking cylinders

is presented with an unmarked ballot, that ballot is locked to the clipboard. When the voter returns to cast their vote, the poll worker checks to ensure that the ballot is still locked to the clipboard and that the paper corner is not torn. Because the clipboard is very noticeable, it is unlikely that a voter would be able to walk out with it.

7.1.1 The election experience

The design of the ballot was approved by GSAÉD a week prior to the election. The GSAÉD election consisted of six contests. Five were positions for office and one was a referendum. Of the five office positions, only one was contested. However the uncontested officials still needed to be confirmed by a majority of voters according to GSAÉD regulations. Thus these four contests consisted of a "yes" or "no" option. The contested office position had two candidates, and the referendum also consisted of a "yes" or "no" option. In essence, the election consisted of six contests, all of which had two candidates/options.

The PunchScan software produced a ballot template and a special XML formatted "drill file", which was used by the machinist to accurately drill the serial number, mark and lock holes in reams of 500 sheets of paper at a time using a computer-positioned vertical mill. We drilled six reams, for a total of 3000 blank ballots. At 19 holes per ballot and 6 reams of paper, the setup and drilling was completed in under half an hour. Given the machine and operator rate of \$75/hour, we were able to drill 3000 PunchScan ballots at a unit cost of \$0.01. Using standard quality white 8.5x11" office paper costing \$0.01 per sheet, the overall unit cost of an unprinted PunchScan ballot was \$0.03 (two sheets of paper, one regular, one drilled).

PunchScan is designed so that all the election data is derived from a key and the key is shared among multiple trustees, ideally trustees who have no incentive to collude with each other (for example, political adversaries). The design of the ballot was approved by GSAÉD , which chose to have a single trustee, the Chief Returning Officer.

The first meeting took place two days before the election; all the commitments were published on the election website immediately. Because it was expected to be a low turnout election, we used the variant with column commitments; we used ten decryption tables, so the probability of cheating and getting away with it was less than one in one thousand. The election was setup with 5000 initial ballots, about half of them being available for the actual election.

A day later, the PunchScan team followed the random selection procedure in [CEA07] which uses volatile stock market data to seed a random number generator to generate the unpredictable choices of which ballot commitments should be opened. Stock data is well-suited for this task because it is unpredictable *a priori*, yet easily verified *a posteriori*. The stock portfolio was comprised of 32 stocks—the subset of the Wired 40 companies traded on NASDAQ.

The pre-election audit took place in two stages. First the CRO entered her passphrase to publish the information about the selected ballots, which consisted of about half of the total 5000 virtual ballots. After this data had been published, the second stage was to actually audit this information against the published commitments from the first meeting. This comparison was performed using an open-source auditing tool provided by the PunchScan team and available from the PunchScan web site. The audit report generated by the program determined that no tampering had taken place. This audit does not require software provided by PunchScan—it follows a publicly-available specification, and can be independently verified by any interested party.

The pre-election audit was performed on the digital representations of the ballots. The ballots that were chosen to be audited were discarded and not used in the election because their unique information was published. The CRO met with the PunchScan team to print 1250 of the remaining 2500 ballots.

Under the supervision of the CRO, the ballots were printed in parallel on five inkjet printers. Three printers were loaned by Hewlett Packard (2 HP-K5400 and 1 HP-K550), and two were provided by the PunchScan team (2 HP-K550). While the printing is not dependent on a particular printer model, the printer needed to satisfy a few requirements: first, the printer should print at the exact positions that it was instructed to (no fit to page, etc), so as to perfectly align the top page of the ballot with the bottom one; second, the printer should be able to print on perforated paper without destroying it; third, the printer should print in color, as the ballots contain color alignment marks. The printer models used for the election were known to meet these restrictions and recommended by the PunchScan team. The printing process took approximately one hour. Upon completion, the ballots were placed into boxes, sealed, and signed along the seal by the CRO.

On the night prior to the election, the PunchScan team hosted a two-hour poll worker training program. For the first hour, an introduction to PunchScan was given explaining some of the theory behind the system, outlining the polling place procedures (this information was also provided in written form), and answering questions.

The second hour provided hands-on experience with the polling place equipment. The poll workers were provided with mock ballots to vote on, and they practiced the procedure of scanning and casting the ballots. We observed an unexpected property of PunchScan: learnability. The poll workers were able to quickly grasp the procedures that they have to follow and moreover, why the procedures were put into place the first time.

The pools were open for the election from the 6th to the 8th of March 2007[2], 10am to 6pm, at five polling places around the campus of the University of Ottawa. A polling station consisted of the following equipment:

- 1 full computer system,
- 1 scanner,
- 1 printer,
- 1 shredder,
- 1 clipboard and lock,
- 1 patterned scissors
- 2 bingo daubers,
- 1 privacy screen.

For this election, the polling station equipment was provided by the PunchScan team to GSAÉD at no charge. However estimates of the cost of purchasing this equipment are as follows: computer ($200[3]), scanner ($100), printer ($200), shredder ($50), clipboard and lock ($20), patterned scissors ($1) daubers ($2) and privacy screen ($10). Thus the marginal cost of a polling place is around $600 if electronic equipment is bought and not rented. While this is a rough estimate, we include it to contrast the cost of purchasing equipment for PunchScan to that of purchasing a DRE. In addition to PunchScan-specific equipment, the usual tables, chairs, power cords and folders were used.

After the completion of the voting process, the scanned receipts were posted on the election website. To transform the receipts, we needed to run a third meeting, where the CRO entered her secret password again. The software was supposed to give out

[2]That entire week, the temperature in Ottawa was around $-25°C$ $(-13°F)$.

[3]Only a basic computer capable of running Java Runtime Environment (JRE) is required.

cleartext ballots, but instead it gave us a very unspecific error. After successive tries, repeated assurances from the CRO that she had not forgotten the secret password, and about 45 minutes of intense uncertainty, we determined that the error was caused because the unrestricted Java policy files for cryptography were not installed on the computer. We switched to a computer that was properly set up, and the CRO was able to immediately announce the winners. The results are summarized in Table 7.1. 154 ballots were cast.

	Contest 1	Contest 2	Contest 3	Contest 4	Contest 5	Referendum
Answer A	118	120	117	121	81	98
Answer B	28	21	31	26	54	43

Table 7.1: GSAÉD election results

The next phase was to check that the transformation of the receipts into cleartext ballots was done correctly. This step took place many days after the election, but it worked without incident, and the audit succeeded.

Poll workers experienced a number of issues. For example, of the 154 ballots cast, only 145 were recorded in the electronic pollbook. More interesting is the poll worker reaction to technological failures. On the software end, there were several instances in which either the polling place software or the electronic pollbook software froze. On the hardware end, there were several instances in which the printers jammed or the wireless internet signal was lost. During these instances we found that all the poll workers undertook to create a handwritten account of the serial number plus a numeric code for each mark (or absence of mark) made by the voter. In some cases the poll workers were not explicitly instructed how to do so. These handwritten records were later recorded electronically, and transcription errors would be subject to detection through the online receipt verification check. This demonstrated an unexpected robustness of PunchScan against a host of denial of service scenarios. Neither the integrity nor the privacy of the election were affected by these failures. Oddly, we did not have ballots that were marked in "interesting" ways. This may be due to the fact that the voters were graduate students, or that the poll workers seemed to understand how the system was working.

7.1.2 Lessons learned

Listed below are some lessons learned from the GSAÉD election:

1. To protect against denial of service attacks, the electronic version of the unfilled cleartext ballots should be recorded on two different permanent media and stored in a secure safe until the elections have been certified and chosen candidates installed in office. This is a safeguard for situations such as forgotten passwords or complete electronic failure.

2. The results of the scanning should be stored in plain text rather than encrypted. Managing the decryption keys may become extremely difficult.

3. The method of mark detection should be improved. In this election, the holes were close together on the top page, and when the voter marked both holes (an over-vote), the marks touched and were wrongfully detected as a single mark.

4. The polling place software should report its status from time to time to a central server, to prevent the need of traveling from one polling place to another in possibly inclement weather.

5. If the OCR of the serial number does not return a valid result, manual correction should be preferred to instead of ballot rescanning.

6. A dolly or a car should be kept available to transport the computer, printer, scanner, shredder, power cords, ballots etc. to the various polling places.

7. The alignment mark should be positioned on the ballot such that, when the ballot is upsidedown, the alignment marks are far away from where they should be, allowing the software the detect that the ballot is upside down.

8. A bootable CD containing all the software needed for the election, ideally an image for a virtual machine, should be used. Installing Java, printer and scanner drivers on the machines used at the polling places can become time consuming. If Linux is needed because of licensing problems, a printer and scanner with Linux drivers should be used.

7.2 The CPSR election

From the beginning of the PunchScan project, our goal was to provide secure and convenient election solutions. Polling place voting is a very good place to start, but the natural next step is remote voting. We have been looking for an opportunity to prove that PunchScan can be used in a remote setting, either by mail-in ballots or fully on-line. But because PunchScan was never designed to be used on-line, we had to come up with new ways that would allow the voters to use their home computers to assist them in casting their votes. This is especially helpful as there have been concerns, in the voting system research community, regarding the manner in which a PunchScan ballot is marked. A computer can make the indirection on the ballot transparent to the voter, and she would only have to click on the desired candidate to cast a vote.

The Computer Professionals for Social Responsibility (CPSR) is a public interest alliance of computer scientists and others concerned about the impact of computer technology on society; it is a global organization promoting the responsible use of computer technology. Lilly Coney has been a collaborator and well-wisher of the PunchScan project; she approached us for help with the CPSR 2007 board member election. The requirements for this election were to allow the voters to get a ballot via the Internet, cast an approval vote and return a privacy-preserving cast ballot, via fax or regular mail.

In the version of PunchScan used for this election, the voter gets an email with a unique unpredictable link to a set of ballots. The voter chooses one of the ballots in PDF form and saves it to her computer. Later, the voter can open it and use the interactive PDF to select her favorite candidates, choose one of the "virtual pages" to keep as a receipt, and print the receipt and mail it or fax it back to PunchScan. The unique link becomes invalid once visited, such that it cannot be visited a second time. We did not allow for the return of the receipt via the Internet because of the following (unlikely) situation: the voter's computer could be infected with a virus that could switch the rightful vote cast by the voter to another candidate. The voter would not know of this change, and, even if she later observed the discrepancy on the public bulletin board, she would have no proof that something went wrong. On the other hand, having a hard copy receipt that she could check and then fax or mail would allow the voter to check the exact information that is sent in, and to have proof if it were incorrectly recorded. We did not protect against improper influence (a person sitting next to the voter while casting a ballot), nor against denial of service attacks. The purpose of this version PunchScan was to increase the integrity of an election mechanism standardly used by CSPR (remote voting) as polling place elections are not practical (the membership of CSPR is spread out across the globe). The improper influence problem is one typical to remote voting. Further, those geographically close to the voters (spouses, relatives, bosses) are probably not interested in the outcome of this particular election; and those who are interested are expected to be physically distanced from the overwhelming majority of the voters and could not be part of an attack related to improper influence. The reason why we did not protect against denial of service attacks was that we did not know how to.

The ballot had two contests, the first being an approval vote for six candidates, and the second one being a runoff vote in case a tie occurred. Details about how the ballot should be handled, along with the fax number and the mail address were also present on the ballot. To treat all ballots uniformly, the PunchScan team arranged a special mail-in address at a postal office in Los Angeles. Whenever something came by mail, the postal employees were instructed to forward it by fax to the PunchScan number. The election ran from August 25 to September 30, 2007. The web site was up and running for the entire period. From the PunchScan perspective, the cost of this specific election was reduced to renting a dedicated mail-in address, about $100 for the entire period of the election. There were 31 valid cast ballots and the results can be viewed in Table 7.2

	Contest 1	Contest 2
Candidate A	9	6
Candidate B	11	5
Candidate C	13	4
Candidate D	21	1
Candidate E	18	2
Candidate F	14	3

Table 7.2: CPSR election results

7.2.1 Lessons learned

This was an uncontrolled election, in the sense that no poll workers were available to instruct the voters on how to mark and handle the ballots, nor did the PunchScan team have any control on the type of software used by the voters to view and mark the ballots. Also, the procedure for returning the ballots was entirely in the hands of the voters.

We experienced multiple difficulties with this election, none great enough to affect the voters or the election. First, the interactive PDFs we created contained JavaScript and worked well on Adobe products (Adobe Reader or Adobe Acrobat). On other PDF viewers, however, the ballots either displayed improperly, or the JavaScript interaction was not as intended. Second, some voters did not read and/or follow the instructions, but simply printed the ballot and circled their favorite candidates (as is the case with some other voter interfaces that the voters might have been familiar with). Other voters followed the instructions, but after printing the receipt, used a pen to either clearly mark some candidates (probably those that were chosen) or to fill in (probably correctly) the letters that were missing. While one may not trust that the voter marked the correct candidates and filled in the correct letters, this is likely; thus voters jeopardized the privacy of their own votes. We were instructed by CPSR to consider any ballot on which the voter intention can be inferred, and we did so. Some of the ballots were invalidated by CPSR due to the lack of signatures on the affidavit attached to the ballot.

Another problem was the scanning software. Voters used a variety of faxes, with various degrees of contrast and luminosity, widely variable resolution and skew. Even though we adapted the software that automatically interpreted the ballots to work with monochrome ballots, the results were poor. Both the OCR and mark detection processes failed multiple times. We had to process the ballots manually and we made mistakes while doing so. Once again, the PunchScan system proved its flexibility: five voters complained that the receipts posted on the PunchScan web site were different from the receipt they held in their hands. In four of the cases the complaint was legitimate and we had made mistakes during manual processing. Before the voting period ended, two separate PunchScan members rechecked all the received ballots against the published receipts. Out of the five complaints, two noticed that their ballot is completely missing from the bulletin board. After some investigation, we discovered that the ballots were sent by mail and were faxed by the post office employees together with other ballots in a single fax transmission, thus producing a single document with multiple pages. Our software was only inspecting the first page.

Overall, the CPSR election was an extremely valuable experience in an uncontrolled environment, where the PunchScan system had the chance to prove its flexibility in the face of the most unexpected errors and faults.

7.3 GW mock election

On November 4th 2008, election day in the US, we organized a mock election at the campus of The George Washington University (GW). The purpose of the election was to informally test the latest front-end, Scantegrity II. We wanted to see if the voters would mark the ballots in uncommon ways, and if they valued the ability to verify the presence of their vote in the virtual ballot box. (It being a mock election, we were able to offer international students the option of expressing their opinions

Name of selection	Totals
Barack Obama/Joe Biden Democrat	149 (79.7%)
John McCain/Sarah Palin Republican	26 (13.9%)
Chuck Baldwin/Darrell L. Castle Constitution	1 (0.5%)
Bob Barr/Wayne A. Root Libertarian	3 (1.6%)
Cynthia A. McKinney/Rosa Clement Green	0 (0.0%)
Ralph Nader/Matt Gonzalez Independent	6 (3.2%)
Ted Weill/Frank McEnulty Reform	1 (0.5%)
No one	0 (0.0%)

Name of selection	Totals
Democrat	98 (52.4%)
Republican	22 (11.8%)
Constitution	0 (0.0%)
Libertarian	11 (5.9%)
Green	4 (2.1%)
Independent	29 (15.5%)
Reform	0 (0.0%)
No one	20 (10.7%)

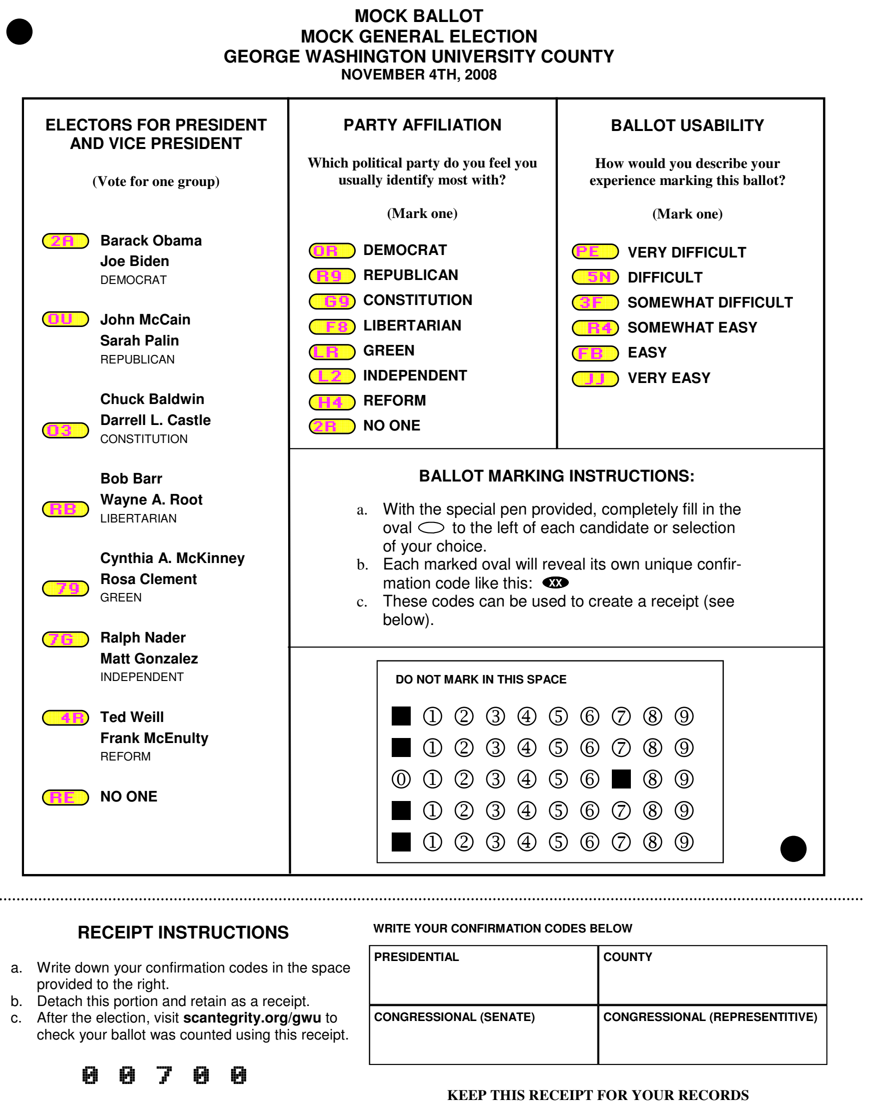
Name of selection	Totals
Very difficult	1 (0.5%)
Difficult	7 (3.7%)
Somewhat difficult	11 (5.9%)
Somewhat easy	16 (8.6%)
Easy	45 (24.1%)
Very easy	102 (54.5%)

Table 7.3: Results of the GW November 4^{th} mock election

on election day.) For this mock election we used the Scantegrity II front-end coupled with the punchscanian mixnet back-end. The same experiment was also replicated by MIT doctoral student Emily Shen, at MIT, on the same day. We printed 200 ballots for the GW election. A sample ballot can be seen in Fig. 7.1. The ballot had three questions, one for president, one for party affiliation and one for ballot usability.

The polls opened at 10am and were scheduled to close at 6pm. Due to unexpected voter participation, we ran out of ballots around 4.30pm and had to close the polling place early. We scanned all the ballots using the copy machine at the computer science department, and processed them using our own detection software. We also obtained scanned ballots from MIT around 5pm. We were able to post results around 7pm, when voters were able to start checking their ballots online. The results of the GW election can be seen in Table 7.3.

It is interesting to see that a majority of the voters considered that it was very easy to mark the ballot, and more than 78% said that it was "easy" or "very easy". This is truly encouraging, since Scantegrity II is one of the first front-ends of end-to-end verifiable voting systems that voters consider easy to use.



MOCK BALLOT
MOCK GENERAL ELECTION
GEORGE WASHINGTON UNIVERSITY COUNTY
NOVEMBER 4TH, 2008

ELECTORS FOR PRESIDENT AND VICE PRESIDENT

(Vote for one group)

2A Barack Obama
Joe Biden
DEMOCRAT

0U John McCain
Sarah Palin
REPUBLICAN

03 Chuck Baldwin
Darrell L. Castle
CONSTITUTION

RB Bob Barr
Wayne A. Root
LIBERTARIAN

79 Cynthia A. McKinney
Rosa Clement
GREEN

7G Ralph Nader
Matt Gonzalez
INDEPENDENT

4R Ted Weill
Frank McEnulty
REFORM

RE NO ONE

PARTY AFFILIATION

Which political party do you feel you usually identify most with?

(Mark one)

0R	DEMOCRAT
R9	REPUBLICAN
G9	CONSTITUTION
F8	LIBERTARIAN
LR	GREEN
L2	INDEPENDENT
H4	REFORM
2R	NO ONE

BALLOT USABILITY

How would you describe your experience marking this ballot?

(Mark one)

PE	VERY DIFFICULT
5N	DIFFICULT
3F	SOMEWHAT DIFFICULT
R4	SOMEWHAT EASY
FB	EASY
JJ	VERY EASY

BALLOT MARKING INSTRUCTIONS:

a. With the special pen provided, completely fill in the oval ⬭ to the left of each candidate or selection of your choice.
b. Each marked oval will reveal its own unique confirmation code like this: XX
c. These codes can be used to create a receipt (see below).

DO NOT MARK IN THIS SPACE

■ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
■ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
⓪ ① ② ③ ④ ⑤ ⑥ ■ ⑧ ⑨
■ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
■ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

RECEIPT INSTRUCTIONS

a. Write down your confirmation codes in the space provided to the right.
b. Detach this portion and retain as a receipt.
c. After the election, visit **scantegrity.org/gwu** to check your ballot was counted using this receipt.

0 0 7 0 0

WRITE YOUR CONFIRMATION CODES BELOW

PRESIDENTIAL	COUNTY
CONGRESSIONAL (SENATE)	CONGRESSIONAL (REPRESENTITIVE)

KEEP THIS RECEIPT FOR YOUR RECORDS

Figure 7.1: Sample ballot for the GW Nov 4^{th}, 2008 mock election

7.3.1 Lessons learned

As with the previous elections held using software we have developed, we learned a number of lessons from this election:

1. We mis-estimated the number of ballots we would need, and ran out of ballots before the polls were to close.
2. The ballot contained some inconsistencies: including the receipt, which contained spaces for four confirmation codes, while the ballot contained only three questions. Also, the use of two different pens, one for marking the ovals, and one for writing the confirmation codes on the receipt, was confusing.
3. Even though the stub voters were to write their conformation codes on was pre-perforated, it was very difficult to detach. The fact that the ballots were printed on cardboard made this even more difficult. We had a voter in a wheelchair who had difficulty holding the pen in his hand to mark the ballot; it was impossible for him to detach the stub.
4. It was generally difficult to explain to people how to vote using Scantegrity II. The best way to address this is to have printed instructions in the voting booth. But we only had one booth, and, on several occasions, many students voting simultaneously. We had to, hence, explain verbally to several voters how to vote; this takes a while and voters often forget.
5. We had problems while scanning approximately 25 of the GW ballots. The alignment mark was too close to the bottom of the ballot, and the scanner only scanned half of it, while the software was looking for a full disc. A simple solution is to leave reasonable margins for the portion of the ballot that gets scanned. Also, if the software cannot interpret the ballot after a reasonable amount of time (e.g. 5 seconds) the scanning software should simply report an error and go to the next ballot.
6. Some voters overvoted; thus they knew two confirmation codes, and could, in theory file a valid complaint. This is a known unaddressed problem with the Scantegrity II front-end. A solution is to have a precinct scanner that refuses to register overvotes.
7. We realized that if somebody walks away with a ballot (without casting it), they can file a valid complaint.

Overall, the GW mock election was a very interesting experiment, with a reasonable number of voters. We were able to see various scenarios and aspects that could be improved.

Chapter 8

Casting assurance for Internet Voting

A great threat to Internet voting is the possibility of a computer virus that spreads to enough computers to elect a winner, regardless of the will of the voters. We present a technique that eliminates the possibility that computer programs alone can meaningfully change votes cast using any computer, with or without malicious software running. The security of our technique depends on the hardness of Captchas (Completely Automated Public Turing Test to Tell Computers and Humans Apart) – problems from artificial intelligence that are hard for computers but easy for humans. One example is the ability of a human to read a distorted text that a computer cannot read. Our technique does not protect against attacks in which humans are involved.

Ensuring that a computer is virus free is virtually impossible. The constant battle between the anti-virus industry and programmers that write malware is well known (the U.S. Computer Emergency Readiness team site is a good example). Computer viruses can go undetected because they can minimally impact computer operation, and can delete themselves immediately after being executed. Operating systems, bootstrapping software, and other essential software can come with a pre-installed virus; such a virus would avoid detection, as it could prevent any anti-virus software from inspecting it. A big obstacle in remote electronic voting (Internet voting) is the ability of a virus to capture the voter choices and harm the computer if the choices are considered "bad" (reboot before casting, delete the hard drive, etc.). This is possible because the virus has some control over the device, and learns the votes while the voter makes her selection. The virus could also change the vote and still present the voter with the "Your vote has been correctly cast" screen, giving the voter a false sense of comfort. This concern has been repetitively raised in reports [JRSW04][KSRW03] and is considered one of the biggest impediments in the adoption of Internet voting (along with denial of service attacks and improper influence). Improper influence refers to a voter voting in the presence of another person, a coercer that exercises pressure in order for the voter to cast a ballot in a certain way. A denial of service is an attempt to make a computer resource unavailable for its rightful users.

We propose a technique that allows a voter to confidently cast a ballot on any computer, without being afraid that a virus or any software can alone (without the help of a human) adversely influence the choices of the voter. When voting in a polling place using stand alone Direct Recording Electronic (DRE) machines, the technique can eliminate the fear that the machines can learn the vote or change it to some specific candidate. We address the problem by associating instances of a Captcha [vABHL03] with the candidates on a ballot. To cast a ballot for her favorite candidate the voter has to solve the Captcha associated with that candidate. We propose that this technique be used in conjunction with an end-to-end, software independent voting system.

The problem we address is viewed as being very difficult. The SERVE report [JRSW04] states that "There really is no good way to build [a secure, all-electronic remote voting system] without a radical change in overall architecture of the Internet and the PC, or some unforeseen security breakthrough". We suggest a technique that takes us one step closer to a solution for such a problem. The technique requires the use of Captchas that are secure enough, and requires the existence of a large number of instances of Captchas. This may be viewed as the cost of the solution to a problem that is widely recognized as being very difficult.

8.1 Security Model

We briefly state the assumptions that our technique is based on. While our assumptions may not hold for any Captcha that is currently in use, it may be that there will exist Captchas that satisfy our assumptions.

1. A Captcha exists. A Captcha is a program that can generate and grade tests that most humans can pass, but current computers cannot. In particular, its security can be based on the hardness of a problem in artificial intelligence for current computers, and depends on the problem being easy for humans. A simple example is the classical Captcha [vABHL03] that is present on popular sites

like Yahoo or Google to prevent, for example, automatic creation of accounts; these Captchas are based on the ability of humans to read distorted text that is automatically generated, while Optical Character Recognition programs cannot correctly read the text.

2. Given a set of Captchas and a solution for one of them, no program can determine, with a non-negligible advantage, which of the Captchas the solution belongs, or does not belong.

3. A voter can solve a Captcha most of the time. When a voter believes that she cannot solve a Captcha, or if the computer that generated it noticed that the solution provided is not valid, the voter gets a new ballot with a new set of Captchas on it.

4. The facility that produces the ballots does not collude (i.e. share more information than strictly needed) with the device used for showing the ballot to the voter. In other words, the computer that is connected to the Internet, and used by the voter to vote, receives only the ballot from the server, and nothing else. If the election authority that produces the electronic ballots is colluding with the malware on the voter's computer, both integrity and privacy can be compromised.

5. The order of the candidates on the ballot is fixed and publicly known.

Our technique does not protect against attacks that involve Captcha farms. A virus can capture the ballot and re-transmit it immediately to people hired to decrypt Captchas and to cast ballots for a certain candidate. In such a scenario, a fairly large number of people would be engaged in the attack, as each Captcha would need to be solved by the human attacker before the connection session expires; thus the adversary has to be faster than the voter. Dan Wallach [Wal08] states that an attack on integrity of an election run via the Internet would take O(1), a single individual; even if Captcha farms are used, our technique improves this estimation to O(P), the number of individuals required to solve the Captchas.

8.2 Description of the technique

We describe a technique that can be used in association with the back-end of Scantegrity, an end-to-end verifiable voting system based on optical scan ballots. We focus

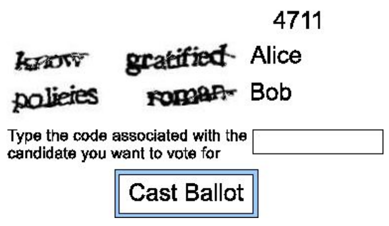

4711
know
gratified
Alice
polieies
roman
Bob
Type the code associated with the candidate you want to vote for
Cast Ballot

Figure 8.1: Scantegrity ballot with Captchas. Captchas compliments of the reCaptcha project - www.captcha.net

on the modification that enables a system such as Scantegrity to be used for remote voting purposes; thus we focus on the casting of the ballot. For a description of the mechanism that allows end-to-end verifiability see Chapter 5. We begin by describing the technique in the context of Scantegrity II. For our work, it is irrelevant how ballots are counted, as our technique addresses only the casting phase.

Each possible vote is associated with a Captcha; Captchas are not repeated within a single ballot. A voter votes by solving the Captcha corresponding to the candidate of her choice. If the assumptions in Sect. 8.1 hold, a virus cannot cast a vote for a particular candidate, because it cannot solve the Captcha associated with it. Other ideas can stand behind the construction of such tests, e.g. recognizing an object or a being in an image, creating a description of a picture, saying if a piece of text makes sense or not, etc.

The voting ceremony proceeds as follows. Using any computer connected to the Internet, the voter accesses the website of the election authority and authenticates herself (the authentication mechanism is out of the scope of our work). The voter is then given a ballot that for each race has a set of Captchas associated with every candidate (see Figure 8.1 for an example of a text based Captcha). For each race, the voter provides the solution to the Captcha (e.g. types in the text that she can read next to her favorite candidate), and presses the cast ballot button. If the voter is visually impaired, she can use an audio Captcha. Thus the technique uses a *cognitive* channel between the voter and the server to provide the voter with the codes corresponding to the individual candidates. If certain AI problems are hard for the computer, this channel cannot be tapped by the computer and hence it cannot (a) determine with certainty what the vote is, and (b) change the vote other than at random. After casting her ballot, the voter receives a signed privacy-preserving receipt from the server, that contains the text that the voter typed in. She keeps the receipt to later check that her vote is correctly posted on the bulletin board.

If the voter is allowed to choose more than one candidate, as in approval or plurality voting, she can simply provide the solution for all the Captchas associated with her favorite candidates. If rank voting is used, the voter can solve the Captchas corresponding to her candidates in order. For range voting, each score that can be assigned to a candidate would have a Captcha associated with it. Our technique does not apply to write-ins.

To determine what the vote is, a virus would have to associate the solution of the Captcha provided by the voter (e.g.the string entered) with the correct candidate. If this problem is hard, and if a program cannot do so, the vote is private. Further, to vote for another specific candidate, a virus would have to correctly guess the solution of the hard AI problem corresponding to the candidate; that is, it would have to break the Captcha, or get the solution by other means.

If the server happens to receive a string that is not in any of the Captchas for that race, it will trigger an alarm and will communicate to the voter that something has gone wrong. The channel used for this alarm can be computer independent (voice, sms, mail etc). Thus, if the Captcha cannot be solved by the virus in the small amount of time that the server allows for a vote using that ballot, no software on the computer can affect the integrity of the election.

8.3 Resistance to Certain Types of Attacks

For text based Captchas, using an alphabet with n symbols and words of length k, there are n^k unique confirmation codes and thus n^k possible Captchas[1]. Assuming a race has x candidates, the probability that a virus correctly guesses the text used in any Captchas on a given ballot is $P = \frac{x}{n^k - 1}$. A practical example is to use words of length eight ($k = 8$) consisting of symbols from an alphabet with thirty two symbols in total ($n = 32$, 26 letters and 6 numerals). For a contest with ten candidates ($x = 10$) the probability that anyone guesses a text that appears on a particular ballot (P) is less than one in one hundred millon (assuming a random guess). Thus, it is very unlikely that a virus can simply guess a valid code. Note that this probability would be larger if it were required that the Captchas represent valid words in the language, as in our example

If the malware on the voters' computers has some knowledge of how the voter is going to vote, it can rearrange the Captchas on the ballot (without knowing what they are). For example, assume the virus has some knowledge that the voter is going

[1]a good Captcha would have all the possibilities equally hard to break and indistinguishable for any software

to vote for Alice, because the voter visited Alice's web site, donated money online, or because it analyzed the content of her inbox. The virus wants to trick the voter into casting a vote for Bob, so it modifies the ballot received from the server by taking the Captcha associated with Bob and putting it next to Alice. To avoid such an attack, a meta Captcha can be constructed: a Captcha that contains Capthcas such that no software can rearrange the individual Captchas. For example, all Captchas in a single race are placed on common meaningful background (e.g a picture of a tree). For text-based Captchas, the malware cannot clip only the text from the Captcha and paste it somewhere else in the background picture, because it does not know the exact boundaries of the text in the Captcha. Trying to clip a regular area (such as a rectangle) is going to perturb the background and the voter will notice it. Similarly, in audio Captchas, a background musical could be used; if pieces are clipped out an rearranged, the background music would sound different. Like in any other voting system where the order of the candidates is fixed, we assume that the order of the candidates is publicly known (perhaps alphabetical). We assume voters would notice if the order of the candidates is different from the official posted order.

A related concern is a forced randomization attack. A coercer can instruct a voter to bring a receipt that is formed in a certain way (e.g. starts with letter A). Like other voter verifiable schemes (e.g. PunchScan, Prêt à Voter), our proposal does not protect against such randomization attacks.

Another attack that we avoid relates to the server that receives the coded votes and knows how the voter voted. It proceeds as follows. The server constructs the Captchas on the ballot, and sends the ballot to the voter. The server then receives the result of the test that corresponds to the chosen candidate (e.g. the text in the Captcha). At this moment, the server knows how the voter voted, and interrupts the protocol, or gives the voter a badly formatted receipt. Thus the voter does not get a receipt, or gets an invalid one. To stop the server from rejecting votes selectively, we use blind signatures [Cha82]. When the voter presses the "Cast Ballot" button, a piece of software (provided by a third party) blinds all the selections and sends them to the server. The server signs the votes, not knowing what they are, and returns them to the client, who now un-blinds them, obtaining a signed receipt of the selection which is sent to the server, in the final step of the protocol. The server cannot reject or change the votes because the voter provides a signed receipt of the vote. Note that the voter now has a receipt with all her selections, signed by the election authority. If she later checks the official website and notices that her selections are not recorded properly, she has her signed receipt, which is an irrefutable proof for the malfeasance.

Unlike other usage of blind signatures, ours does not require an anonymous channel; all the communication takes place in a single session and is authenticated. This is because, in our scheme, the vote is encoded.

The voter needs to be sure that the ballot she receives is properly constructed by the server (in [CCC$^+$08] this is referred to as the printing audit). This is done using a cut and choose protocol. The voter may choose to spoil the received ballot and the server would then have to publicly reveal all the information with regards to that ballot. There are two ways this can be implemented. First, the server can simply sign all the ballots that it hands out. If it turns out that the ballot is not constructed properly, the voter has irrefutable proof (the signed blank ballot) of the malfeasance. Another way of solving this problem is that the server hands out to the voter an unsigned ballot, the voter decides to spoil the ballot and sends it to the server using a blind signature protocol. The server has to sign the ballot because it does not know if it signs a vote or a spoiled ballot. The server may authenticate itself to the voter via a technique proposed in [SV07], which is an alternative to digital signatures and can be checked without access to trusted computation.

When digital signatures are used on a potential infected computer, the malware can interfere with the verification of the signature. In the context of our solution, this would disrupt the voting process, equivalent to a denial of service attack, but with the focus on the client, rather than the traditional denial of service attack directed to the server. We assume the only scope of the virus is to either find out how to voter voted or to change the vote. Our solution does not address denial of service attacks, which can be mounted in a variety of ways without infecting hosts.

As mentioned in Sect. 8.6, the proposed technique does not address the possibility of a human coercer, whether sitting next to the voter, or remote. It addresses only mass coercion by a virus that can go undetected while infecting most of the voting hosts. It also does not address the instance when humans are used to solve Captchas and to change the vote.

8.4 Accessibility

Visually impaired voters can use an audio version of a Captcha. A test is delivered to them via voice, and they have to solve the test to prove that they are humans and not software. Many popular web sites (www.gmail.com, www.hotmail.com) use audio Captchas in addition to visual ones, for accessibility reasons. Some attacks have been targeted specifically at audio Captchas [Win], with a higher rate of success than with their visual equivalent. It is possible that this is due to the fact that visual Captchas have received considerably more attention than have audio Captchas, and that more research in the design of audio Captchas will result in more secure audio Captchas. Further, audio and visual Captchas can be designed around hard problems in language processing, with the same hard AI problem – summarizing whether a given sentence is about “rain” or about “snow” for example – represented either visually or in audio for a voter, based on the voter’s preference.

8.4.1 Illiterate voters

We describe a simple Captcha-based technique, inspired by PunchScan, that can be used by illiterate voters. We assume that the order of the candidates is fixed and that the voter can recognize her favorite candidate. The voter can memorize the way the candidate is written, or the candidate can have a graphical representation, e.g. a symbol or a color. With each candidate is associated a symbol that cannot be interpreted by a computer program, but can easily be interpreted by a human voter, e.g. a picture of a dog. At the bottom of each contest there is a big picture containing symbols that represent the same concepts as the symbols associated with the candidates. To vote for a particular candidate, the voter has to select the symbol on the picture that represents the same concept that is represented by the symbol next to her favorite candidate. The symbols, their association with candidates, and the locus in which they appear on the bottom picture are all uniformly random.

We give a simple concrete example of such a ballot: there are two candidates on a ballot, Alice and Bob. Alice has a picture of a black cat associated with her and Bob of a brown dog. At the bottom of the ballot there is a picture containing a white cat and a red dog. To cast a vote for Alice, the voter would have to click on the white cat. The assumption is that a computer program cannot distinguish a picture of a cat from the picture of a dog, nor can it detect where the symbols are placed on the bottom picture.

8.5 Captcha Security and the privacy and integrity of the voting system

Unlike some hard problems in computer science, scientists agree that several AI problems will eventually be solved. However, we do not require Captchas to be secure forever for the system to be secure against attacks on the integrity of the vote tally. If the Captcha cannot be broken in the short amount of time of a voting session, then the device cannot figure out how the voter voted, and cannot meaningfully change the vote; at most it can change the vote at random. The receipt that the voter gets is a record that the vote was properly recorded and thus is a proof of integrity. On the other hand, if Captchas are broken in a longer time frame, then a computer virus can figure out how the voter voted, breaching privacy, but not directly affecting the integrity of the election.

Recently, there appeared some attacks [YAb], [YAa] that solve with high probability some forms of visual Captchas based on the user recognizing the distorted text on an image. Other attacks have been aimed at audio Captchas [Win]. At the same time, new forms of tests that are claimed to be solvable only by humans have also appeared [ali]. We urge the reader to treat the specific Captchas used by us only as examples of the manner in which Captchas will be used. We expect that any implementation of our system will use the most secure Captchas available at the time an election is run.

8.6 Potential attacks

The purpose of our work is to stop a team of computer programmers from creating a virus that would manipulate an election. We achieve this by preventing the voting machine from learning the vote while the voter is using it (and possibly from ever learning the vote). Our technique does not guard against a human looking at the votes. The human can sit behind the voter, or can remotely receive the voted ballot from the voting device. Such attack may have scaling problems, since a large group of people needs to be involved to coerce or modify (in real time) a significant fraction of all votes.

The voter can take a screen shot of the ballot and bring it to a coercer. If the coercer is a human, our technique does not protect against such attacks. Two observations are worth making: first, the screen shot can be altered by the voter to produce a convincing proof to the coercer, and second, this attack is valid for any voting system, be it remote (as in mail-in voting) or precinct-based (taking a movie of yourself voting).

It is worth mentioning that a man-in-the-middle attack does not make any sense, if run by a computer program and not a human, that is, a machine-in-the-middle attack. The machine-in-the-middle gets the ballot from the server and gives the voter another ballot (produced by the adversary). The machine-in-the-middle would then learn how the voter intends to vote, but cannot do anything about it. It cannot change the vote, nor can it transmit it unaltered to the server (because it cannot read the Captcha on the real ballot). Thus this attack would not stop the voter from voting, because the server would allow another ballot to be cast later on, since the first session was unsuccessful from the point of view of the server.

Chapter 9

Conclusions

The development of electronic voting in recent years had led to more single points of failure in the vote casting and counting process. While it has been noted, about forty years ago [Ber69], that computers cannot be trusted when it comes to election technology, the most widely used voting systems of 2008 still use technology that is neither voter-verifiable nor universally verifiable. While the fundamental law of the United States allows each citizen to freely cast a secret ballot, it is silent about how the voting system must assure the voters that their votes were correctly counted. It is essential to have security mechanisms in place that would stop those counting the ballots from deciding who the winner of an election is, without regard to the will of the voters.

Secure voting methods have been devised for the past twenty-seven years. Most of these systems were devised for voting remotely, assuming the voter had access to trusted computation. More recently, there have been new voter-verifiable voting systems that allow the voter to check that her vote has been correctly included in the virtual ballot box, without compromising her privacy, and without requiring her to have access to trusted computation while voting. During the past four years, the scientific community has begun to agree on the advantages offered by this new paradigm, and several designs and implementations have contributed to the advancement of the field.

This dissertation has described some of the first research in voter-verifiable system design and implementation. It has made the following contributions:

First, it has proposed a general framework for secure electronic voting systems that are based on mixes, contributing to a better understanding and evaluation of the general properties offered by the techniques that recent mixnet-based voting systems implement. Using the principle of decoupling, the current work takes a fresh look at voting systems that have originally been proposed as monoliths and separates the voter-verifiable component of these systems, named the front-end, from the universally-verifiable component, named the back-end. The front-end is responsible for the interaction with the voters, ensuring that all the votes are recorded as intended. The back-end deals with transforming the receipts the voters receive into cleartext ballots, in a manner that is universally-verifiable. Both parts preserve the secrecy of the vote.

By decomposing mixnet based voting systems into two parts, a family of voting systems that share the same parts is now available. Being able to combine any of the front-ends with any of the back-ends gives enormous flexibility in the construction of systems. Some front-ends may be easier to use, others may provide better privacy, or be compatible with the existing voting equipment. Some back-ends are very efficient and provide election results in minutes, while others are completely decentralized and provide better reliability. By knowing the properties of the front and back-ends, election officials possess a pallet of choices, and can find the combination that best suits the needs of their voters and their budget.

Second, this dissertation has provided mathematical formalism and proofs of the security properties (integrity and privacy) of both the front and back-ends. While some of the proofs are general and refer to entire classes of the two components (e.g. symmetric ballots vs. asymmetric ballots), other demonstrations are particular for the specific component of the voting system (e.g. the integrity assurance that the PunchScan back-end provides every time it is audited).

Third, this dissertation describes the development of two components that were developed while working towards this dissertation: the Scantegrity and Scantegrity II front-ends as well as the pointer-based back-end. The advantages of these new voting system components are their simplicity and their compatibility with voting equipment that has already been deployed and used in real elections. They do not attempt to substitute the underlying optical scan of commonly used voting systems in the USA, but build a layer of security on top of it.

Fourth, all the front and back-ends have been implemented and tested in real or mock elections. One cannot overstate the value of building and running elections using abstract descriptions of voting systems. A series of practical details have been clarified, this work in turn led to the proposal of new techniques that addressed previously-overlooked aspects. Real elections validate the importance of techniques such as contest partitioning, provably secure and private audits, recovery from failed audits, separate page generating and printing, as well as the influence of the configuration parameters on the security and privacy provided by the system. A performance analysis has been performed, and the results proved that these new systems are several orders of magnitude faster than the previously known general techniques.

Finally, this thesis presents a general technique for building a new front-end that allows the voters to cast a ballot using a computer that is potentially infected with malware, ensuring that their vote would not be manipulated, nor their choice revealed to the machine. This technique should help in making the inevitable, Internet voting, more secure and resistant to attacks that were previously thought to be unstoppable.

9.1 Future work

Future work can focus on improving individual front and back-ends that are considered unsuitable for use in certain scenarios. For example, an implementation of a front-end that is accessible to voters with disabilities is an immediate goal. Such a front-end may not assume that it can avail of the write-once nature of paper, and hence may not derive security from the write-once media. New variants of back-ends are already being developed. For example, the Aperio [ECA08] back-end is easier to explain to the general public.

Once a system is developed for voters that are visually impaired, it can be used by all voters if it eliminates the use of paper in the polling place. It is still a research question if a front-end with strong properties of incoercibility and integrity can be built without the use of paper.

Many of the existing systems concentrate the information—possessed by the voting system, and required for the decryption of encrypted ballots and the ensuing tally—in a single entity. While, in theory, the entity can be distributed and can exist only at given moments in time, when it comes to manufacturing/printing the ballots, a single point of trust is created. Future work can look for clever ways to distribute such hot spots.

More research is needed for generating and evaluating good user interface designs that present voters with an experience that feels familiar and easy. How the ballots looks and how it is marked is the single most important aspect of the entire voter experience.

It is interesting to explore how this work can be applied by organizations that monitor the elections internationally. Big international organizations like OSCE and

UN, or U.S. organizations like IFES and PEW can now supervise if the collected ballots are correctly tabulated with digital observers that do not have to travel to the country where the election is held.

It is not currently clear if the end-to-end voting systems are certifiable under the current regulations. The constitutions of the U.S. states guarantee a private vote and a secret ballot, but are silent about the integrity of the tally. One option is to investigate the compatibility with current lawa and regulation. The second option is to investigate the possibility of updating the regulation to include the two components of end-to-end systems. In this case, interesting questions arise. Can a singe component be certified or only a voting system in its totality? What are the performance requirements for each functional component? Are components from two different voting system completely interchangeable?

Another avenue is formal proofs of correctness for each component presented in this thesis. While the philosophy behind the end-to-end system is that of there is a bug in the implementation, it will be revealed by the data that the protocol produces, having a formal modeling of the protocol and formally proving its correctness, possibly using automated tools, would certainly help.

The construction of an open specification and open source election management system is an engineering exercise that should not be underestimated. The ballots in the U.S. contain questions for all government levels: federal, state, regional and local. The voters are voting for a large number of office, from the U.S. president to the neighborhood cat catcher and mosquito control. Should the eahc entity have a back-end of its own to tally all of their the questions (i.e. the federal election authority has a back-end that tallies the question for U.S. president, the state authority tallies the question for senator, etc)? Or should each entity that runs an election have its own independent back-end? While the first option offers more privacy (because the poll of encrypted votes is larger), it does not allow for having local results reported for federal races. Such tradeoffs should be carefully considered and such management systems should be built in the very near future.

But the most important aspect of the future work is the education of the election officials and of the public at large about the existence of theoretical techniques, and their practical implementations, that empower voters to check that their own vote is recorded as cast, and allow everyone to check that all the votes were counted as recorded. The existing techniques are only as good as the benefits they provide to the voters. If left unused, such techniques will fail to contribute to the advancement of the field and to the transparency of democracy in general.

References

[ABE25] Masayuki ABE. Universally verifiable mix-net with verification work independent of the number of mix-servers. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 83(7):1431–1440, 20000725.

[Adi06] Ben Adida. *Advances in Cryptographic Voting Systems*. PhD thesis, MIT, 2006.

[Adi08] Ben Adida. Helios: Web-based open audit voting. In *Proceedings of the Fourteenth USENIX Security Symposium (USENIX Security 2008)*. Usenix, July 2008.

[ali] alipr.com. Next-generation captcha exploits the semantic gap. `http://tech.slashdot.org/article.pl?sid=08/04/23/0044223` [Online; accessed 2-June-2008].

[AR06] Ben Adida and Ronald L. Rivest. Scratch & Vote: self-contained paper-based cryptographic voting. In *WPES '06: Proceedings of the 5th ACM workshop on Privacy in electronic society*, pages 29–40, New York, NY, USA, 2006. ACM Press.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[Bel] Mary Bellis. The history of voting machines. `http://inventors.about.com/library/weekly/aa111300b.htm?once=true&`.

[Ben87] Josh Cohen Benaloh. *Verifiable Secret Ballot Elections*. PhD thesis, Yale University, 1987.

[Ben08] Josh Benaloh. Administrative and public verifiability: Can we have both? In *IAVoSS Workshop On Trustworthy Elections (WOTE 2006)*, KU Leuven, belgium, July 2008.

[Ber69] Richard Bergholz. How election can be rigged via computers. *Los Angeles Times*, July 1969.

[BF78] Steven Brams and Pete Fishburn. Approval voting. In *American Political Science Review*, volume 72, pages 831–847, 1978.

[Bin06] Mark Binker. Printers failed on voting machines, 2006.

[BJK$^+$] Josh Benaloh, Douglas Jones, John Kelsey, Eric Lazarus, Ronald Rivest, and Paul Miller. Judges evaluation of the punchscan system. `http://vocomp.org/evaluation.php`.

[BJR01] Shuki Bruck, David Jefferson, and Ronald L. Rivest. A modular voting architecture ("Frogs"). In *WOTE01 Workshop on Trustworthy Elections*, 2001.

[BM] Erika Bolstad and Curtis Morgan. Gambling vote glitch mars tally in broward county. `http://www.ejfi.org/Voting/Voting-121.htm#gambling` [Online; accessed 12-August-2008].

[Bow] Debra Bowen. California secretary of state, voting systems review: Top-to-bottom review. Website. `http://www.sos.ca.gov/elections/elections_vsr.htm`.

[Bra] The Brad Blog. Website. `http://www.bradblog.com/`.

[BT94] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 544–553, New York, NY, USA, 1994. ACM.

[BY86] Josh C Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters. In *PODC '86: Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, pages 52–62, New York, NY, USA, 1986. ACM.

[CCC$^+$08] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity ii: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology on USENIX/Accurate Electronic Voting Technology Workshop*. USENIX Association, 2008.

[CEA07] Jeremy Clark, Aleks Essex, and Carlisle Adams. Secure and observable auditing of electronic voting systems using stock indices. In *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2007.

[CEC$^+$08] David Chaum, Aleks Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan T. Sherman, and Poorvi Vora. Scantegrity: End-to-end voter verifiable optical-scan voting. *IEEE Security and Privacy*, May/June 2008.

[Cel04] RABA Inovative Solution Cell. Trusted agent reportdiebold accuvote-ts voting system (raba). `http://www.raba.com/press/TA_Report_AccuVote.pdf`, January 20 2004. Technical Report.

[CFSY96] Ronald Cramer, Matthew Franklin, Berry Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. *Lecture Notes in Computer Science*, 1070:72–83, 1996.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *Lecture Notes in Computer Science*, 1233:103–??, 1997.

[Cha81] David L. Chaum. Untraceable electronic mail, return address, and digital pseudonym. *Communication of ACM*, February 1981.

[Cha82] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto'82*, 1982.

[Cha88a] D. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *J. Cryptol.*, 1(1):65–75, 1988.

[Cha88b] D. Chaum. Elections with unconditionally-secret ballots and disruption equivalent to breaking rsa. In *Lecture Notes in Computer Science on Advances in Cryptology-EUROCRYPT'88*, pages 177–182, New York, NY, USA, 1988. Springer-Verlag New York, Inc.

[Cha04] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security and Privacy*, pages 38–47, January/February 2004.

[Cha05a] David Chaum. Punchscan - invioted talk during the advanced crypto course, December 2005.

[Cha05b] David Chaum. Recent results in electronic voting. In *Presentation at Frontiers in Electronic Elections (FEE 2005)*, Milan, Italy, September 2005. ECRYPT and ESORICS.

[CHVW05] Lillie Coney, Joseph L. Hall, Poorvi L. Vora, and David Wagner. Towards a privacy measurement criterion for voting systems. In *National Conference on Digital Government Research*, May 2005.

[CM08] Jessie Carpenter (City Clerk) and Barbara Matthews (City Manager). Worksession - takoma park. `http://www.takomaparkmd.gov/clerk/agenda/items/2008/063008-5.pdf`, June 2008.

[Com07] U.S. Election Assistance Commission. Uniform and overseas citizens absentee voting act - survey report findings, 2007.

[CRS05] David Chaum, Peter Y. A. Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In *In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, ESORICS,* *volume 3679 of Lecture Notes in Computer Science*, pages 118–139. Springer, 2005.

[CvdGRV07] David Chaum, Jeroen van de Graaf, Peter Y. A. Ryan, and Poorvi L. Vora. Secret ballot elections with unconditional integrity. Technical report, IACR Eprint, 2007. `http://eprint.iacr.org/` or `http://www.seas.gwu.edu/~poorvi/cgrv2007.pdf`.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 581–596, London, UK, 2002. Springer-Verlag.

[Dun] John Dunbar. Internet voting project cost pentagon 73,809usd per vote. Website. `http://www.publicintegrity.org/telecom/report.aspx?aid=297`.

[ECA08] Aleks Essex, Jeremy Clark, and Carlisle Adams. Aperio: High integrity elections for developing countries. In *IAVoSS Workshop On Trustworthy Elections (WOTE 2008)*, Katholieke Universiteit, Leuven, Belgium, July 2008.

[Ele05] Election Assistance Commission. Voluntary voting system guidelines. `http://www.eac.gov/votingsystems/voting-system-certification/2005-vvsg/?searchterm=vvsg`, 2005.

[Ele07] Election Assistance Commission. Voluntary voting system guidelines, 2007.

[FCS06] Kevin Fisher, Richard Carback, and Alan T. Sherman. Punchscan: Introduction and system definition of a high-integrity election system. In *IAVoSS Workshop On Trustworthy Elections (WOTE 2006)*, Robinson College, Cambridge UK, June 2006. `www.wote2006.org`.

[FHF06] Ariel J. Feldman, Alex J. Halderman, and Edward W. Felten. Security analysis of the diebold accuvote-ts voting machine. Technical report, Princeton University, September 2006.

[FOO93] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques*, pages 244–251, London, UK, 1993. Springer-Verlag.

[FS01] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 368–387, London, UK, 2001. Springer-Verlag.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[Ger01] Ed Gerck. Voting with witness voting. In *WOTE*, 2001.

[Gid05] John Gideon. Diebold's failure in california, 2005. `http://www.votetrustusa.org/index.php?option=com_content&task=view&id=96&Itemid=30`.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, 1991.

[Har] Bev Haris. Black box voting. Website. `http://www.bbvforums.org`.

[HBO06] HBO. Hacking democracy - movie, 2006.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[HV08] Ben Hosp and Poorvi L. Vora. An information-theoretic model of voting systems. `http://www.seas.gwu.edu/~poorvi/VotingModel.pdf`, Oct/Nov 2008.

[IPSC07] Richard T. Carback III, Stefan Popoveniuc, Alan T. Sherman, and David Chaum. Punchscan with independent ballot sheets: Simplifying ballot printing and distribution with independently selected ballot halves. In *IAVoSS Workshop On Trustworthy Elections (WOTE 2007)*, University of Ottawa, Canada, June 2007.

[JJ99] Markus Jakobsson and Ari Juels. Millimix: Mixing in small batches. Technical Report 99-33, DIMACS, 10 1999.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proceedings of the 11th USENIX Security Symposium*, pages 339–353, Berkeley, CA, USA, 2002. USENIX Association.

[JL97] Juang and Lei. A secure and practical electronic voting scheme for real world environments. *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, 1997.

[Jon03] D. W. Jones. A brief illustrated history of voting. http://www.cs.uiowa.edu/~jones/voting/pictures/, 2003.

[JR07] Rui Joaquim and Carlos Ribeiro. Codevoting: protecting against malicious vote manipulation at the voter's PC, 2007. http://kathrin.dagstuhl.de/files/Submissions/07/07311/07311.JoaquimRui2.ExtAbstract!.pdf [Online; accessed 19-October-2007].

[JRSW04] David Jefferson, Aviel D. Rubin, Barbara Simons, and David Wagner. A security analysis of the secure electronic registration and voting experiment (SERVE). http://www.servesecurityreport.org/, January 21 2004. Technical Report.

[KKW06] Aggelos Kiayias, Michael Korman, and David Walluck. An internet voting system supporting user privacy. *Computer Security Applications Conference, 2006. ACSAC '06. 22nd Annual*, pages 165–174, Dec. 2006.

[KM06] Thad Kousser and Megan Mullin. Will vote-by-mail elections increase participation? evidence from california counties. Technical report, UCSD, 2006. http://weber.ucsd.edu/~tkousser/WillVote-by-MailElectionsIncreaseTurnout.pdf.

[KRMC07] John Kelsey, Andrew RegenScheid, Tal Moran, and David Chaum. Some random attacks on paper-based e2e systems, 2007. http://kathrin.dagstuhl.de/files/Materials/07/07311/07311.KelseyJohn.Slides.pdf [Online; accessed 17-November-2008].

[KSRW03] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting machine. Technical Report TR-2003-19, Johns Hopkins Information Security Institute, July 23 2003.

[Lan] Gentry Lange. King county, WA, 13 days of vote counting, Seattle special election, 2007. Website. http://novbm.wordpress.com/2007/04/07/king-countywa-13-days-of-vote-counting-seattle-special-election-2007/.

[LK00] B. Lee and K. Kim. Receipt-free electronic voting through collaboration of voter and honest verifier, 2000.

[LV01] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 14(4):255–293, 2001.

[Mer00] Rebecca Mercuri. *Electronic Vote Tabulation Checks and Balances.* PhD thesis, University of Pennsylvania, Philadelphia, PA., October 2000.

[MH96] Michels and Horster. Some remarks on a receipt-free and universally verifiable mix-type voting scheme. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag, 1996.

[MN07] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privac. `http://www.wisdom.weizmann.ac.il/~talm/papers/MN06-voting.pdf`, June 2007.

[Mor04] Michelle Morgante. E-voting bug plagued san diego county, 2004. `http://www.verifiedvotingfoundation.org/article.php?id=1536` [Online; accessed 12-August-2008].

[MRCM08] Stephen Chong Michael R. Clarkson and Andrew C. Myers. Civitas: Toward a secure voting system. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2008.

[MZK$^+$74] Richard H. McKay, Paul G. Ziebold, James D. Kirby, Douglas R. Hetzel, and James U. Snydacker. U.s. patent 3,793,505 - electronic voting machine, 1974.

[Nef01a] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *8th ACM Conference on Computer and Communications Security*, pages 116–125, 2001.

[Nef01b] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125, New York, NY, USA, 2001. ACM.

[New02] Dallas Morning News. Touch screen voting machines lose unknown number of votes in dallas, 2002. `http://www.ejfi.org/Voting/Voting-111.htm#dallas`.

[NS95] Moni Naor and Adi Shamir. Visual cryptography. *Lecture Notes in Computer Science LNCS*, 950:1–12, 1995.

[OKST97] W Ogata, K Kurosawa, K Sako, and K Takatani. Fault tolerant anonymous channel. In *Information and Communications Security — First International Conference*, volume 1334, pages 440–444, Beijing, China, 11–14 1997. Springer-Verlag.

[oS07] University of Surrey. The prêt à voter battle bus, 2007. `http://www.vocomp.org/teams.php` [Online; accessed 19-October-2007].

[oT07] Wroclaw University of Technology. The voting ducks voting system, 2007. `http://www.vocomp.org/teams.php` [Online; accessed 19-October-2007].

[PCE$^+$08] Stefan Popoveniuc, Heremy Clark, Aleks Essex, Richard Carback, and David Chaum. Securing optical scan voting. *Frontiers of Electronic voting - to appear in Springer LNCS*, 2008.

[PER03] Nathanael Paul, David Evans, and Avi Rubin. Authentication for remote voting. In *Workshop on Human-Computer Interaction and Security Systems*, Fort Lauderdale, Florida, April 2003. `http://www.cs.rice.edu/~dwallach/pub/remote-voting2003.pdf`.

[PEW08] PEW. Election 2008 in review. `http://www.pewtrusts.org/uploadedFiles/wwwpewtrustsorg/Reports/Election_reform/ElectionInReviewPDF%2520Final.pdf`, 2008.

[PH06] Stefan Popoveniuc and Ben Hosp. An introduction to PunchScan. In *IAVoSS Workshop On Trustworthy Elections (WOTE 2006)*, Robinson College, Cambridge UK, June 2006.

[PIK94] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *EUROCRYPT '93: Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 248–259, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[PL08] Stefan Popoveniuc and David Lundin. A simple technique for safely using punchscan and pret a voter in mail-in elections. In *VOTE-ID*, pages 150–155, 2008.

[pol07] Election Website for Polk County, Florida. `http://www.polkelections.com/`, October 2007.

[PP90] Birgit Pfitzmann and Andreas Pfitzmann. How to break the direct RSA-implementation of MIXes. *Lecture Notes in Computer Science*, 434:373–??, 1990.

[Pro01] Caltech-MIT Voting Technology Project. Voting what is, what could be. `http://www.vote.caltech.edu/`, July 2001.

[PV] Stefan Popoveniuc and Poorvi Vora. A framework for secure electronic voting. *Cryptologia.* accepted for publication.

[PV08a] Stefan Popoveniuc and Poorvi Vora. A framework for secure electronic voting. In *IAVoSS Workshop On Trustworthy Elections (WOTE 2008)*, Katholieke Universiteit, Leuven, Belgium, July 2008.

[PV08b] Stefan Popoveniuc and Poorvi Vora. Remote ballot casting with captchas. In *3rd Workshop on information and System Security*, 2008.

[Rad95] Michael J. Radwin. An untraceable, universally verifiable voting scheme. `http://www.radwin.org/michael/projects/voting.html`, 12 1995.

[RH] Dan Rather-HDNet. The trouble with touchscreens - the movie. http://video.google.com/videoplay?docid=8424269587813976191&q=dan+rather+reports&total=84&start=0&num=10&so=0&type=search&plindex=1 [Online; accessed 19-October-2007].

[Riv06] Ronald L. Rivest. The ThreeBallot voting system. http://theory.lcs.mit.edu/~rivest/, October 2006.

[RS07] Ronald L. Rivest and Warren D. Smith. Three voting protocols: Three-ballot, vav, and twin. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology on USENIX/Accurate Electronic Voting Technology Workshop*, pages 16–16, Berkeley, CA, USA, 2007. USENIX Association.

[RW] Ron Rivest and John Wack. On the notion of "software independence" in voting systems. http://vote.nist.gov/SI-in-voting.pdf DRAFT Version Retrieved on September 25, 2007.

[Sak07] Kazue Sako. On svis project. http://kathrin.dagstuhl.de/files/Materials/07/07311/07311.SakoKazue.Slides.ppt, 2007.

[Sal75] Roy G. Saltman. Effective use of computer technology in vote-tallying. Technical report, NIST, 1975.

[Sal06] Roy G. Saltman. *The History and Politics of Voting Technology*. Palgrave Macmillan, 2006.

[Sal07] Roy Saltman. Improving the voting process: A multi-disciplinary and politicized problem. http://www.wisdom.weizmann.ac.il/~talm/papers/MN06-voting.pdf, September 2007.

[Sch99] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. *Lecture Notes in Computer Science*, 1666:148–164, 1999.

[Ser06] Election Data Services. 69 million voters will use optical scan ballots in 2006, 2006. http://www.electiondataservices.com/EDSInc_VEStudy2006.pdf.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[SK] Warren D. Smith and Jan Kok. Center for range voting. Website http://rangevoting.org/.

[SK94] Kazue Sako and Joe Kilian. Secure voting using partially compatible homomorphisms. In *Advances in Cryptology - CRYPTO'94*, pages 411–424, 1994.

[SK07] Warren D. Smith and Jan Kok. Plurality voting, 2007. `http://rangevoting.org/Plurality.html` [Online; accesed 20-October-2007].

[Smi07] Warren D. Smith. Ants, bees, and computers agree range voting is best single-winner system, 2007. `http://www.math.temple.edu/~wds/homepage/naturebees.pdf` [Online; accesed 20-October-2007].

[SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984.

[SV07] Rahul Simha and Poorvi L. Vora. Vote verification using captcha like primitives. In *IAVoSS Workshop On Trustworthy Elections (WOTE 2007)*, University of Ottawa, Ottawa, Canada, June 2007.

[tea] The Auburn University team. Prime 3, one mahcine, one vote for everyone. `http://www.primevotingsystem.com/` [Online; accessed 19-October-2007].

[Tur36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[Uni94] International Telecommunication Union. X.200 informaiton technology - open systems interconnection - basic reference model, 1994.

[VAB$^+$04] P. L. Vora, B. Adida, R. Bucholz, D. Chaum, D. L. Dill, D. Jefferson, D. W. Jones, W. Lattin, A. D. Rubin, M. I. Shamos, and M. Yung. Evaluation of voting systems. *Comm. of the ACM*, Nov. 2004.

[vABHL03] L. von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard AI problems for security. In *Proceedings of Eurocrypt*, pages 294–311, 2003.

[Ver] Verified voting. Website. `http://www.verifiedvoting.org/`.

[Vor05] Poorvi L. Vora. David Chaum's voter verification using encrypted paper receipts. Cryptology ePrint Archive, Report 2005/050, 2005. `http://eprint.iacr.org/`.

[Vot] Vote trust usa. Website. `http://www.votetrustusa.org/`.

[Wal08] Dan Wallach. Voting system risk assessment via computational complexity analysis. *William and Mary Bill of Rights Journal*, December 2008.

[Wika] Wikipedia. Instant-runoff voting — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Instant-runoff_voting` [Online; accessed 11-November-2007].

[Wikb] Wikipedia. Ostracon — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Ostracon` [Online; accessed 19-October-2007].

[Wikc] Wikipedia. Papal conclave — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Papal_conclave` [Online; accessed 19-October-2007].

[Wikd] Wikipedia. Preferential voting — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Preferential_voting` [Online; accessed 11-November-2007].

[Wike] Wikipedia. Secret ballot — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Secret_ballot` [Online; accessed 09-November-2008].

[Wikf] Wikipedia. Volusia error — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Volusia_error` [Online; accessed 19-October-2007].

[Win] WintercoreLabs. Breaking gmails audio captcha. `http://blog.wintercore.com/?p=11` [Online; accessed 2-June-2008].

[YAa] Jeff Yan and Ahmad Salah El Ahmad. Is cheap labour behind the scene? - low-cost automated attacks on yahoo captchas. `http://homepages.cs.ncl.ac.uk/jeff.yan/yahoo.htm` [Online; accessed 2-June-2008].

[YAb] Jeff Yan and Ahmad Salah El Ahmad. A low-cost attack on a microsoft captcha. `http://homepages.cs.ncl.ac.uk/jeff.yan/msn_draft.pdf` [Online; accessed 2-June-2008].

[Yao82] Andrew Yao. Protocols for secure computations. *Proc. 23d IEEE Symp. on Foundations of Computer Science*, pages 160–164, 1982.

[Zul25] T David Zulkerman. The voting machine, 1925.

Chapter 10

APPENDIX A - Proofs

Theorem 1. Let n be the total number of ballots, and k the number of ballots that are incorrectly formed. If f ballots are chosen at random in step 4, the probability of not detecting any of the k incorrectly formed ballots is smaller than $min[(1-\frac{f}{n})^k, (1-\frac{k}{n})^f]$.

Proof. We will first prove a lemma

Lemma 20. *In a box there are n balls, k black and the rest white. Uniformly at random, f balls are extracted without replacement and their color is inspected. The probability that all f extracted balls are white is lower than or equal to $min[(1-\frac{f}{n})^k, (1-\frac{k}{n})^f]$.*

Proof. The number of ways to choose f balls out of the n balls in the box is $\begin{pmatrix} n \\ f \end{pmatrix}$ (n choose f). The number of ways to choose f white balls from a total of n balls where k of them are black, is $\begin{pmatrix} n-k \\ f \end{pmatrix}$ (choose f white balls out of the total $n-k$ that are white). The probability that all the f extracted ballots are all white is the number of favorable cases divided by the number of total cases:

$$p = \frac{\begin{pmatrix} n-k \\ f \end{pmatrix}}{\begin{pmatrix} n \\ f \end{pmatrix}} = \frac{\frac{(n-k)!}{f!(n-k-f)!}}{\frac{n!}{f!(n-f)!}} = \frac{\frac{(n-k)!}{(n-k-f)!}}{\frac{n!}{(n-f)!}}$$

Note that $f+k \leq n$, so that $n-k-f \geq 0$ and $(n-k-f)!$ exists. If $f+k > n$ then the probability is 0.

We compute two upper bounds on the above probability:

$$\begin{aligned}
\frac{(n-k)!}{n!} \times \frac{(n-f)!}{(n-k-f)!} &= \frac{(n-f)\times(n-f-1)\times ...(n-f-k+1)}{n\times(n-1)\times ...\times(n-k+1)} \\
&= \frac{n-f}{n} \times \frac{n-f-1}{n-1} \times ... \times \frac{n-f-k+1}{n-k+1} \\
&= (1-\frac{f}{n}) \times (1-\frac{f}{n-1}) \times ... \times (1-\frac{f}{n-k+1}) \\
&\leq (1-\frac{f}{n})^k
\end{aligned}$$

$$\begin{aligned}
\frac{(n-k)!}{(n-k-f)!} \times \frac{(n-f)!}{n!} &= \frac{(n-k)\times(n-k-1)\times ...\times(n-k-f+1)}{n\times(n-1)\times ...\times(n-f+1)} \\
&= \frac{n-k}{n} \times \frac{n-k-1}{n-1} \times ... \times \frac{n-k-f+1}{n-f+1} \\
&= (1-\frac{k}{n}) \times (1-\frac{k}{n-1}) \times ... \times (1-\frac{k}{n-f+1}) \\
&\leq (1-\frac{k}{n})^f
\end{aligned}$$

It follows that $p \leq min[(1-\frac{f}{n})^k, (1-\frac{k}{n})^f]$. □

Having lemma 20 the proof for Theorem 1 follows directly. □

Corollary 2. Given R, $0 < R < 1$, and $k \geq 1$, then the values of n and f can be chosen such that Pr[the one inconsistent ballots is not detected] $\leq R$.

Proof. Taking $k = 1$ in the inequalities of Theorem 1, we get $p_1 \leq 1 - \frac{f}{n}$, therefore $1 - \frac{f}{n} \leq R \Leftrightarrow \frac{f}{n} \geq 1 - R$.

Because f is strictly greater than zero and strictly less than n then $\lim_{n\to\infty} \lim_{f\to n} \frac{f}{n} = 1$ and thus we can always choose n and $f < n$ such that $\frac{f}{n}$ is as close to 1 as desired; having $R > 0$ concludes the demonstration. As smaller values of f are required to detect the presence of $k > 1$ inconsistent ballots, $f \geq n(1-R)$ will always suffice to find an inconsistent ballot with probability at least $1 - R$, for any number cheated on.

□

Theorem 2. If the back-end does not compute $D_1(i, w)$ or $D_2 \circ D_1(i, w)$ correctly for k ballots in step 11, the probability that all the records on the SPBB are consistent at the end of the protocol is $(\frac{1}{2})^k$.

Proof. In step 11, the back-end can guess, for a particular ballot transformation, which side (D_1 or D_2) the auditor will ask to see in step 12. It would then compute the other transformation incorrectly, and would be detected with probability $\frac{1}{2}$. Because the choices of the auditor in step 12 are independent, the probability that the back-end guesses correctly on any one ballot is independent of the probability of guessing correctly for any other ballot. Thus if the back-end cheats on k ballots, the probability that this does not get exposed through step 12 is $(\frac{1}{2})^k$. $\square$

Lemma 8 Suppose, in step 12, the unpredictable choice requires that the commitment to f_i be opened for q values of i, $1 < q < p - 1$, where p is the number of cast ballots. Then the privacy of any of the q corresponding ballots, as defined in definition 5, is q, and that of any of the other ballots is $q - p$.

Proof. Let $\{f_i\}_{i \in \mathcal{D} \subset \mathcal{B}}$ be the commitments opened. Using lemma 5 with D_1 and D_2 instead of T and B, we conclude that, for any of the q ballots in $\mathcal{D}$, for which $D_1(i, \_)$ is revealed, a function $D'_2(\pi(i), \_)$ can be constructed such that $D'_2 \circ D_1(i, w) = (j, v)$ (where w is the encrypted vote created by $\mathcal{F}(i, v) = (i, w)$ and v is the cleartext vote) for any j such that $j \neq \pi_2 \circ \pi_1(i')$ for $i' \in \bar{\mathcal{D}}$, where $\bar{\mathcal{D}}$ denotes the complement of $\mathcal{D}$. The size of this set is exactly q, and thus the size of the privacy set for any of the q ballots in $\mathcal{D}$ is q.

Since step 12 says that for all $D_1((i, \_)$ that are not revealed, it follows that $D_2(\pi_1(i), \_)$ are revealed; thus the cardinality of the last set is $p - q$, proving the lemma. $\square$

Theorem 11. For a PunchScan back-end with d linked or unlinked $\mathcal{D}s$, row-wise commitments and n cast ballots, assume that there exist k ballots for which back-end did not compute $D_2 \circ D_1(i, v)$ correctly. Then the probability that the records on the SPBB after the protocol finished are consistent is $(\frac{1}{2^k})^d$.

Proof. Theorem 2 says that, for a single $\mathcal{D}$ the back-end would have made k correct guesses. When d $\mathcal{D}s$ are available, and they are independent from an integrity point of view, then the probabilities multiple $(\frac{1}{2})^k \times (\frac{1}{2})^k \times ... \times (\frac{1}{2})^k$, in total d factors for a cumulative probability of $(\frac{1}{2^k})^d$. $\square$

Theorem 12. For a PunchScan back-end with d linked or unlinked $\mathcal{D}s$, column-wise commitments and n cast ballots, assume that there exist k ballots for which the back-end did not compute $D_2 \circ D_1$ correctly. Then the probability that the records on the SPBB are consistent at the end of the protocol is $(\frac{1}{2})^d$.

Proof. We assume that it is known ahead of time that the column-wise commitments are used and that either D_1^i or D_2^i are revealed in step 12, but never both. Then there is a single random choice for each $\mathcal{D}^i$ and thus the probability that the back-end cheated on the transformation that is not opened is $\frac{1}{2}$. When d $\mathcal{D}s$ are available, and they are independent from an integrity point of view, then the probabilities multiply $(\frac{1}{2}) \times (\frac{1}{2}) \times ... \times (\frac{1}{2})$, in total d factors for a cumulative probability of $(\frac{1}{2})^d$. $\square$

Theorem 13. For a PunchScan back-end with d linked or unlinked $\mathcal{D}s$, column-wise commitments and n cast ballots, after the protocol finished the privacy of any of the ballots is n.

Proof. We choose an arbitrary encrypted ballot (i, w) and an arbitrary $\mathcal{D}^i$. Assume that for the particular $\mathcal{D}^i$, D_1^i was completely revealed. From lemma 5 it follows that D_2^i can be constructed such that it translates the encrypted ballot (i, w) into any cleartext ballot (j, v). This is possible because no information about D_2^i was revealed and revealing D_1^i does not reveal any information about D_2^i. It follows that the privacy of ballot (i, w) is n, the total number of cleartext ballots. The same reasoning can be applied if D_2^i is revealed and D_1^i is hidden.

Note that it does not matter if $\mathcal{D}s$ are linked or unlinked. $\square$

Theorem 14. Assume in an election there are c candidates and each candidate receives n_i votes, $\sum_{i=1}^{c} n_i = n$. The number of ways to partition the set of votes into two subsets each with the same distribution of votes, and to choose one of the tallies, is $\frac{2^n}{\sqrt{\prod_{i=1}^{c} n_i}}$

Proof. We will first prove a lemma

Lemma 21. *x choose $\frac{x}{2}$=$\begin{pmatrix} x \\ \frac{x}{2} \end{pmatrix}$ is approximately $\frac{2^{x-1}}{\sqrt{x}}$.*

Proof. Stirling's approximation states that $x! \approx x^x e^{-x}\sqrt{2\pi x}$

Thus, $\begin{pmatrix} x \\ \frac{x}{2} \end{pmatrix} = \frac{x!}{\frac{x}{2}!(x-\frac{x}{2})!} = \frac{x!}{\frac{x}{2}!\frac{x}{2}!} = \frac{x!}{(\frac{x}{2}!)^2} \approx \frac{x^x e^{-x}\sqrt{2\pi x}}{(\frac{x}{2}^{\frac{x}{2}} e^{-\frac{x}{2}}\sqrt{2\pi\frac{x}{2}})^2} = \frac{x^x e^{-x}\sqrt{2\pi x}}{(\frac{x}{2})^x e^{-x} 2\pi\frac{x}{2}} = \frac{x^x\sqrt{2\pi x}}{(\frac{x}{2})^x 2\pi\frac{x}{2}} = 2\frac{x^x\sqrt{2\pi x}}{(\frac{x}{2})^x 2\pi x} =$

$2\frac{x^x}{(\frac{x}{2})^x\sqrt{2\pi x}} = 2\frac{1}{\sqrt{2\pi x}}\frac{x^x}{(\frac{x}{2})^x} = 2\frac{1}{\sqrt{2\pi x}}(\frac{x}{\frac{x}{2}})^x = 2\frac{1}{\sqrt{2\pi x}}2^x = \frac{2}{\sqrt{2\pi}}\frac{2^x}{\sqrt{x}} \approx 0.8\frac{2^x}{\sqrt{x}} = 0.8 \times 2 \times \frac{2^{x-1}}{\sqrt{x}} \approx \frac{2^{x-1}}{\sqrt{x}}$ □

Now go onto proving Theorem 14. Lemma 21 says that the number of ways to divide n_i identical votes into two is approximately $\frac{2^{n_i-1}}{\sqrt{n_i}}$. If we aggregate this result for all candidates, we get $\prod_{i=1}^{c}\frac{2^{n_i-1}}{\sqrt{n_i}} = \frac{2^{\sum_{i=1}^{c} n_i-1}}{\sqrt{\prod_{i=1}^{c} n_i}} = \frac{2^{n-c}}{\sqrt{\prod_{i=1}^{c} n_i}}$. To correct for all the possible ways these half parts can be associated, we have to multiple by 2^c. The final number of possible combinations is $\frac{2^n}{\sqrt{\prod_{i=1}^{c} n_i}}$. □

Theorem 15. A PunchScan back-end has information theoretical integrity if and only if it uses a perfectly binding commitment scheme.

Proof. First we prove that that a perfectly binding commitment scheme is necessary to get information theoretical integrity, using proof by contradiction. Assume that the back-end is computationally unbounded, and a commitment scheme that is not perfectly binding is used. Then the back-end can open the commitments in at least two ways (if not more). This means that, in step 12 of the protocol, the back-end can open either D_1 or D_2 in at least two different ways that would satisfy the auditor. But the two ways could be used to link to two different outputs, only one of which would be the correct one. A computationally unbounded back-end could hence cheat the auditor into believing that a particular output was correctly processed when it was not.

Now we prove that a perfectly binding commitment scheme is sufficient to get information theoretical integrity. Assume that a perfectly binding commitment scheme is used; even if the back-end is computationally unbounded, it can only open the commitment in one unique way. The integrity guarantees of the protocol have been proven by the theorems in Sect. 5.2 without making any assumptions about the powers of the EA, so the back-end could be computationally unbounded and the proofs would still be valid. Thus it is sufficient that a perfectly binding commitment scheme is used. □

Theorem 17. Assuming a symmetric ballot with p parts, let k be the number of ballots misprinted out of the total of n cast ballots. If r voters check their receipt, then the probability of not detecting any of the misprinted ballots is

$$\sum_{x=0}^{k}(1-\frac{1}{p})^x \left(\begin{array}{c} r \\ x \end{array}\right) \prod_{i=0}^{x-1}\frac{k-i}{n-i}\prod_{i=0}^{r-x-1}\frac{n-k-i}{n-x-i} \tag{10.1}$$

Proof. We will first prove a lemma.

Lemma 22. . *Assuming a symmetric ballot with p parts is used, let r be the number of unique voters that check for printing correctness. Assume k out of the r checked ballots have been incorrectly printed. Then the probability that none of the k out of r misprinted ballots are detected is $(1 - \frac{1}{p})^k$.*

Proof. For each ballot that the r voters check, the voter chose at random one of its parts, each part containing an equal amount of information. A symmetric ballot is considered printed incorrectly if at least one part is incorrectly printed. It does not make any sense for the cheater to misprint more that one part, since it is sufficient for his scope and it minimizes the probability of getting caught. Thus, for a single ballot, the probability that the ballot was printed incorrectly and the voter did not choose the misprinted part is $1-\frac{1}{p}$. Assuming that the probability of a ballot i being printed correctly is independent of the probability of ballot $j \neq i$ being printed correctly, the probability that none of the k misprinted ballots in step 5 are detected when r ballots are checked by the voters is $(1 - \frac{1}{p})^k$. □

We now prove Theorem 17. Assume exactly x of the r verified ballots are misprinted. Fixing a particular order of the verified ballots, assume that the first x ballots are misprinted. The probability of this to happen is $\prod_{i=0}^{x-1} \frac{k-i}{n-i} \prod_{i=0}^{r-x-1} \frac{n-k-i}{n-x-i}$ (the first x ballots have to be misprinted and the rest of the ballots have to be printed correctly). From lemma 22 the probability of not catching x misprinted ballots is $(1 - \frac{1}{p})^x$. To eliminate the fact that only the first x verified ballots can be misprinted (the ordering constraint) and to consider that any x of the r can be misprinted we have to correct with a factor of $\left(\begin{array}{c} r \\ x \end{array} \right)$. Thus the probability that there are exactly x out of the r verified ballots that are misprinted and none of them is detected is $(1 - \frac{1}{p})^x \times \left(\begin{array}{c} x \\ r \end{array} \right) \prod_{i=0}^{x-1} \frac{k-i}{n-i} \prod_{i=0}^{r-x-1} \frac{n-k-i}{n-x-i}$.

We have to sum this probability from 0 to k, since it may happen that none of the misprinted ballots are verified, or one is, or two are, or all the misprinted ballots are verified. Thus it follows that the probability that none of the k misprinted ballots are detected by the r voters that check is $\sum_{x=0}^{k}(1-\frac{1}{p})^x \left(\begin{array}{c} x \\ r \end{array} \right) \prod_{i=0}^{x-1} \frac{k-i}{n-i} \prod_{i=0}^{r-x-1} \frac{n-k-i}{n-x-i}$ □

Theorem 19 The receipt created by the PunchScan front-end does not reveal anything more than the number of candidates the voter chose.

Proof. Because $\mathcal{F}(i, \emptyset) = (i, 0)$ it is obvious that if the voter marked less than the maximum number allowed, this is visible in the public record when inspecting $\mathcal{F}(i, v)$.

Assume the voter voted for the maximum number of candidates allowed and that she choose $T(i, \_)$ to keep, for a particular i. From lemma 18 it results that $B(i, \_)$ can be created such that $\mathcal{F}(i, v)$ represents any $v \in V^*$. Because the $B(i, \_)$ committed to in step 3 is not revealed and $B(i, \_)$ is independent of any $B(j, \_)$ revealed in step 4 and no other information is revealed up to this step, it follows that no information is revealed about the vote v that the voter chose.

The proof is the same if the candidate choose $B(i, \_)$ as her receipt. $\square$

Chapter 11

Appendix B: An Intuitive Description of Pointer-Based Mixnets

The simplification of the general and PunchScan mixnets comes from a very simple observation: in voting systems, the number of cleartext messages that comes out of the mixnet is very limited, usually equal to the number of candidates on the ballot. If an election has n ballots and each ballot has c candidates, then the output can be presented to have c distinct messages that can come from n source. All the outputs are initially not flagged and a cast vote for a particular candidate causes a particular output message to be flagged.

The output is a set of $n \times c$ possible messages, grouped by candidates. The input of the mixnet is also a set of $n \times c$ possible messages, grouped by ballots. When a vote is cast, an input message is flagged. After all the votes are cast, the mixnet flags the corresponding outputs, and the clear votes flagged in the output can be counted by anyone.

Another view can be that of a big decryption mechanism. The input is a message of length n, and each symbol in the message can take c values. The pointer based mixnet is transforming this "encrypted" message into a cleartext message of the same length and of the same form. This observation comes from the fact that any encryption can be viewed as a permutation and instead of doing encryption/decryption at each stage of the traditional mixnets, the entire mixnet itself is the decryption.

11.0.1 A physical model

We present a physical model of the pointer based back-end that is equivalent to the software model. A technical reader can skip to Sect. 5.3.1. For exemplification we use a ballot with one contest and two candidates, Alice and Bob. The number of voters in this election is four, thus the number of ballots is four. The physical model does not scale well to a longer ballot or to more voters, but it is sufficient to introduce all the necessary notions for the computerized model, which scales to an election that uses a normal size ballot and millions of voters.

There are three tables: one red, one yellow and one green. The red table is the highest, at 7 feet above the ground, the yellow is in the middle, at 4 feet above the ground and the green table is the lowest at 1 foot above ground. The red and the green table are connected with plastic hoses and the hoses go through the yellow table. See Fig. 11.1.

The red table has oval regions cut in it, and in each oval region there is an oval piece of wood. The green table has trapezoid shapes cut into it, and in each hole there is a trapezoid piece of wood. The yellow table has holes cut into it in the shape of diamond (rhombus), and in each hole there is a piece of wood that is rhombus in shape. There is only one way a trapezoid piece of wood can fit into a trapezoid hole. An oval piece of wood or a rhombus can be put into their respective hole in two ways (at 0 degrees or rotated at 180 degrees).

There are four holes and four pieces of wood of each color (and thus of each shape and for each table). One oval is connected to one rhombus with two hoses and the rhombus is further connected to the trapezoid with two other hoses. The hoses in any pair are initial parallel (they do not cross). When all this setup is installed, it looks like three tables one on top of the other, with pairs of hoses coming from the top table, through the middle table and to the bottom table, all the hoses being parallel. In other words, the leftmost hose on the top table is the leftmost hose on the middle table and the leftmost hose on the bottom table. Same goes for the second left hose and for all the hoses.

The ovals represent the ballots. The two hoses that are connected to an oval on the highest table correspond to the two candidates, Alice and Bob. The holes where the hoses are connected are labeled with Alice and respectively Bob. The names are carved into wood. When all the hoses are parallel, the holes on each oval have carved first Alice and then Bob into then. Initially, anybody is invited to check that the order is first Alice and then Bob on all ovals and that all the hoses are parallel.

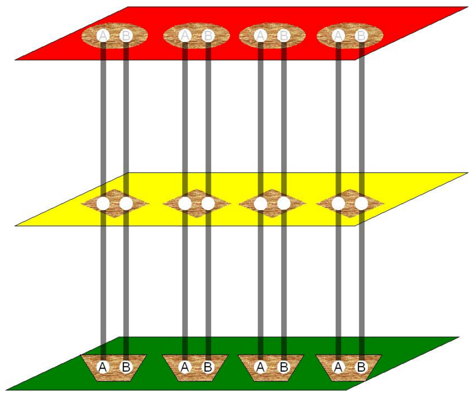


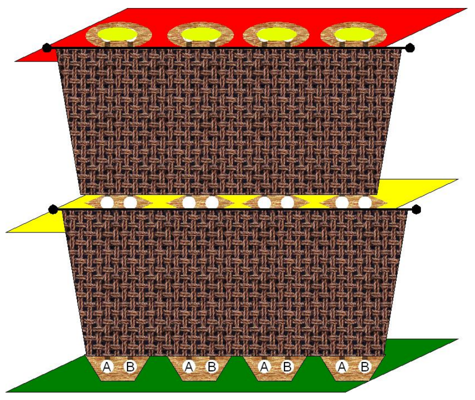
Figure 11.1: A voting system with hoses and water, initial setup: three tables on top of each other, four ballots, two candidates.

There are two operations that are possible: rotations by 180 degrees and switches. On the red table, the ovals are only allowed to be rotated. On the yellow table, the rhombuses are allowed to be both rotated and switched with another rhombus. On the blue table, the trapezoids are only allowed to be switched. It is not possible to rotate a trapezoid, since it would not fit the hole anymore. Also, it is not possible to switch an oval with another oval on the top table (although this rule is actually irrelevant).

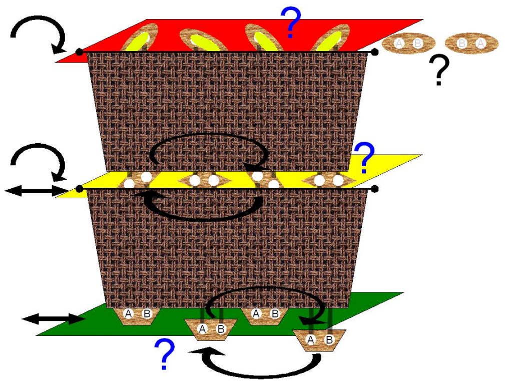
After the initial check is done, the following things happen:

- The names carved into the ovals are covered with wax.
- The space between the top table and the middle table is covered with a curtain, such that only the ends of the hoses are visible, but their body is covered. Same for the space between the middle and low table. See Fig. 11.2(a)
- The people running the election (or simply anyone) are invited one by one to do any number of the three possible operations: rotate an oval, switch two trapezoids or switch and rotate two diamonds. The operations are done in private, i.e. no one except the person doing the operation is allowed to see what operations are done. See Fig. 11.2(b).

Because of the two curtains, no one can know after the switches and rotations which end of the hose connected to the red ovals corresponds to which end of the hose connected to the yellow diamonds. Same for diamonds and trapezoids. This is because the trapezoid and the rhombuses have been shuffled among them. Also, because the



(a) Hoses are covered with curtains. Ballots are covered with wax



(b) The ovals are rotated. The diamonds are rotated and switched. The trapezoids are switched.

Figure 11.2: A voting system with hoses and water: preparing for an election

ovals have been rotated, no one knows anymore if, for a particular oval, Alice is on the right or Bob is on the right. See Fig. 11.2(b).

The curtains are going to always cover the body of the hoses and are never to be removed. Instead of curtains, a brick wall can be built around the body of the hoses. However, the ends of the hoses are always visible.

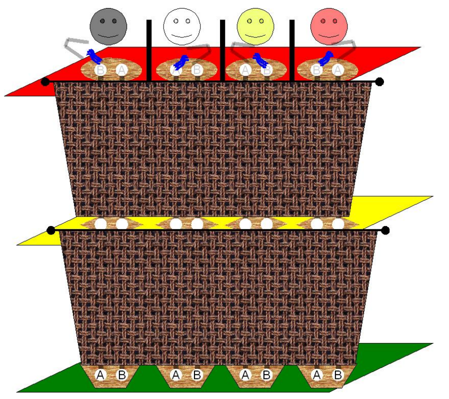
The voting ceremony is as follows (see Fig. 11.3(a)):

1. after proper identification a voter is given a 12oz glass full of water. The voter is allowed to approach the red table and to choose one of the red ovals to vote on;

2. the voter scratches off the wax and sees the names of the two candidates. There are two possibilities, either the voter sees Alice on the left and Bob on the right, or Bob on the left and Alice on the right. No one else except the voter knows this order.

3. the voter pours the water from the glass into the hole that corresponds to her favorite candidate. Everyone sees where the water is poured, but no one except the voter knows to which candidate the water gets poured into, since the ovals have been arbitrarily rotated by many people at the beginning;

4. the voter covers the entire red oval with wax, such that the names of the candidates become invisible. Also the wax goes to the edge of the oval, such that the oval cannot be rotated anymore.


(a) Each voter is given a glass of water. In the privacy of the both, the voter scratches off the wax, sees the order of the candidates. The voters pour water on the favorite candidate

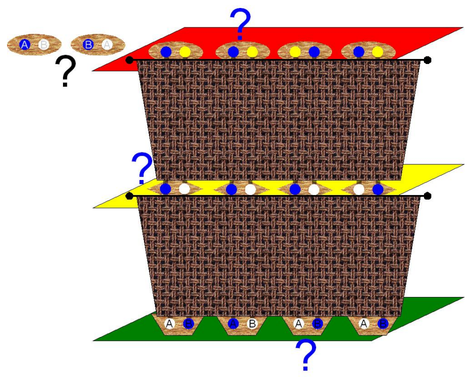

(b) The water is left to flow. Voters can check that the water is where they poured it. Results are computable by anyone by looking at the trapezoids. Votes are not linked with Results

Figure 11.3: A voting system with hoses and water: voting secretly

5. using a steel stylus, the voter signs on the wax;

There is a mechanism that can hold the water from flowing through the hoses from the higher table to the middle tables. Same for the middle table. After all the voters vote, the mechanism that holds the water at the level of the red table is put into action. 8oz of water are allowed to flow from each oval, all of them at the exact same time. The water flows down to the middle table, where it stops. Because the rhombuses on the middle table were switched, there is no way to know that a particular water poured into a particular rhombus.

Then the mechanism that holds the water at the middle level is put into action and 4oz of water are allowed to flow from each rhombus, all of them at the exact same time. The water flows down to the blue table, where it stops. Again, because the trapezoids were switched, no one can know in which particular trapezoid the water from any particular rhombus ended up (see Fig. 11.3(b)).

The trapezoid cannot be rotated (because they would not fit their hole anymore). At the beginning, before the ovals were rotated, all the hoses were parallel. That means that, on each trapezoid, the water in the left hose indicates a vote for Alice and the water from the right hose a vote for Bob. Thus vote counting is easily done by anyone, by counting how many left hoses have water and attributing that many votes to Alice. Same for Bob and the right hoses with water.

To make sure that everything went smoothly, an audit procedure is performed.

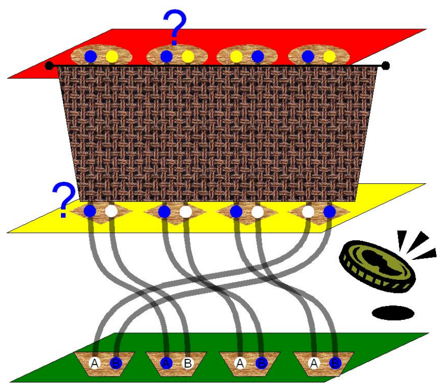


Figure 11.4: A voting system with hoses and water: auditing the transformations

Each voter can make sure that the water that she poured is still there. If she believes it is not there, she can check the signature she made on the wax with the steel stylus, and if the signature is not there anymore, she has proof that someone tampered with her vote.

An unbiased coin is flipped. If the coin comes up heads, then the bottom curtain is removed (if tails, the top curtain is removed). Everyone can check that both ends of each hose have water and that the hoses run in pairs. If anyone was able to fiddle with the setting, this audit would catch any cheating with a probability of 50%. The audit does not reveal how anyone voted, since the top curtain covers the connections between the ovals and the rhombuses. So even after the audit, any oval could be connected with any rhombus (see Fig. 11.4).

If the 50% probability of cheating is too low, a different kind of audit can be performed: when looking at the yellow (middle) table, an unbiased coin is flipped for each rhombus:

- if the coin comes up heads, then a red fluorescent gas that is lighter than air is injected into the hose with a syringe. The gas travels up the hose and must end up in one of ovals that contains water.
- if the coin comes up tails, then a blue fluorescent gas that is heavier than air is injected into the hose with a syringe. The gas travels down the hose and must end up in one of the trapezoids that contains water.

This second audit would catch a cheater that managed to fiddle with all the ballots with probability $1 - \frac{1}{2^4} = 1 - \frac{1}{16} = 93.7\%$ while still preserving the unlinkability between ovals and trapezoids. If the audit needs to be repeated, the gas is let out, the ovals are left untouched, the rhombuses rotated and scrambled, and the trapezoids are scrambled. Then the audit is repeated.