ABSTRACT

Title of Thesis: Security Innovations in the Punchscan Voting System
Richard T. Carback III, Master of Science, 2008

Thesis directed by: Alan T. Sherman, Associate Professor
Department of Computer Science and Electrical Engineering

Punchscan is a so-called *end-to-end (E2E)* voting system that uses cryptographic techniques to give each voter or observer an unprecedented degree of confidence in voting results while preserving the secret ballot. It is unique because it uses a special dual-layer paper ballot that permits each voter to make choices in secret, preventing a machine or person from learning or changing the choices on the ballot at the polling place.

Unfortunately, while Punchscan provides significant integrity properties it does not prevent choices from being discovered if someone sees the ballot before the voter, or if a central workstation is compromised. We propose two improvements that mitigate these privacy problems:

1. We design a trusted workstation that distributes trust across multiple trustees. Each trustee brings a separate read-only copy of approved election software and compares it to copies brought by other trustees. Chosen at random after all copies are verified by each trustee, the software manages a secret sharing scheme that prevents any trustee from performing election functions without a minimum number of other trustees present.

2. Currently, Punchscan can print only ballot layers together. We modify it to treat each ballot layer separately, allowing each to be combined independently by a voter in the voting booth. This permits more efficient distribution strategies that protect voter privacy.

Keywords. Punchscan, voting system security, optical scan, end-to-end cryptographic audit trail.

Security Innovations in the Punchscan Voting System

by

Richard T. Carback III

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
2008

To my family: I would have never gone this far without their advice and enouragement.

ACKNOWLEDGMENTS

Alan Sherman, my advisor, provided significant helpful advice and feedback. He found funding to keep me working at UMBC over the summers. Most importantly, conversations with him have given me invaluable insights to almost everything I can imagine—from what it takes to be a successful researcher to long term investment strategies. Over the course of writing this document, my writing style and strategy has improved so much that I do not even recognize some documents that I have written anymore, and I owe this improvement to his persistence.

David Chaum, my long distance mentor and collaborator from California, is an endless source of ideas and the creator of the Punchscan system. He is the source of the implementation strategy behind the independent ballot sheets modification to Punchscan by suggesting that the decryption process need only decrypt the discarded sheet to work. He also certified the validity of my user contributed secret sharing system being a new idea, and provided useful feedback on the software validation technique.

Aleksander Essex, Jeremy Clark, Stefan Popoveniuc, Ben Hosp, and Jeremy Robin, who are my other long distance collaborators, were a highly available resource for me to consult and were instrumental in testing the software I built during the course of this research. Without Aleksander, Stefan, and Jeremy Clark, I do not believe the Punchscan project could have gone on to win VoComp. Aleksander and Jeremy created a verification program, and Stefan created all of the scanning and cryptographic software in Punchscan. All of them were instrumental to the student election our project carried out at the University of Ottawa.

Kevin Fisher and John Krautheim, my lab mates, provided helpful discussions when I had problems. Kevin Fisher, who was an early collaborator to the Punchscan project,

worked with me to create a concrete conceptualization of what the Punchscan system would look like if it were implemented and used in a real election in the months following David's first talk about it at UMBC. He also first talked about the idea to treat the ballot sheets in Punchscan independently.

I received support over the summer to work on security lab exercises through the Department of Defense Information Assurance Scholarship Program (DoD IASP). UMBC supported me with a teaching assistantship during the school year, and my experiences as a teaching assistant have made me a better student, teacher, and researcher. NSF supported the VoComp competition, which I competed in with Jeremy, Aleksander, and Stefan.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

1. **CNC**. Cut and choose protocol, see 2.2.7.
2. **CSPRNG**. Cryptographically Secure Pseudo Random Number Generator.
3. **DRE**. Direct Recording Electronic voting system.
4. **E2E**. Systems that support end-to-end cryptographic independent verification, see 2.3.
5. **EA**. Election Authority. A small, trusted group of election officials.
6. **EAC**. Election Assistance Commission.
7. **IBS**. Independent Ballot Sheets, a modification of the original Punchscan voting system.
8. **ITA**. Independent Testing Authority. An organization that verifies software does what it claims to do and provides an independently packaged version of the tested software.
9. **PAV**. The Prêt à Voter voting system.
10. **SSS**. Secret Sharing Scheme, see 2.2.6.
11. **UCSS**. User Contributed Secret Sharing.
12. **USBCS**. Unconditionally Secure Bit Commitment Scheme.

Chapter 1

INTRODUCTION

Punchscan is a novel voting system that lets each voter use a special paper ballot. The voter can take home part of this special paper ballot, and use it to verify that her choices were properly received by making sure it appears on an official record posted by election officials. By itself, the piece of the ballot that the voter takes home, an encrypted privacy-preserving receipt, does not contain enough information to identify any choices made on the ballot and this prevents a voter from proving to others how she voted.

After all receipts have been posted to the public record, a small group of election officials, the election trustees, must meet to decrypt the receipts. Anyone can be involved with a mandatory audit mechanism that ensures that all votes are counted properly. Through the privacy-preserving receipt and audit process Punchscan offers unprecedented levels of integrity, privacy, and transparency throughout the election process.

By contrast, the direct recording electronic (DRE) and optical scan systems used in today's U.S. elections do not provide a receipt or a mandatory audit process. Instead, they provide audit logs and an optional a paper record, but these records can be modified without physical, procedural, and software enforced security protections.

Punchscan is an ambitious system with a design and set of requirements that are the first of their kind. Thus, there are many things that we can learn from the study of this

system. During the course of refining and developing Punchscan, we created a specialized trusted workstation and expanded Punchscan to support what we call independent ballot sheets, making it more flexible to deploy in a real environment.

1.1 Motivation

In December 2005, David Chaum presented the Punchscan ballot and counting protocol at UMBC, but he did not specify how it might be implemented in practice. The protocol made a number of assumptions—such as the presence of a trusted workstation, a trusted printer, a binding commitment scheme, a public bulletin board system, paper folding and printing systems, etc—that needed to be addressed for practical use of the system.

During the course of the development and implementation of Punchscan, we found two issues for which there were no obvious solutions: reliance on trusted components for some properties and unspecified key management. In our work, we investigated these issues and provided solutions that prevent compromise of the properties attained by the original protocol. We presented Punchscan at the intercollegiate voting systems competition (VoComp) and won first place [22] among the competing voting systems.

1.2 Overview of Thesis

Chapter 2 provides necessary background information and an overview of voting systems related to Punchscan. Chapter 3 is a detailed overview of the Punchscan voting system.

Chapter 4 outlines the requirements and proposes an architecture for Punchscan's trusted workstation This workstation is trusted with masking ballot id numbers from the final results in a punchscan election, because it must generate the secret encryption key necessary to generate and decrypt the ballots and perform the actions that decrypt the bal-

lots. We propose a workstation that addresses the general problem of multiple users trusting the integrity and privacy of computations done on one machine.

Our workstation design presents a protocol for multiparty initialization of a computer and a secret sharing system that generates its secret from user contributed entropy. The multiparty initialization routine involves each user bringing their own copy of software to check every other user's copy. The secret sharing system lets each user contribute entropy to a master key which is stored in a way that allows a subset of members to decrypt it when all members are not present.

Chapter 5 presents an improved modification of Punchscan that permits the two sheet ballot to be printed at different printers and combined at the polling place. It also provides an overview of a similar modification and explains a flaw that breaks its privacy.

This modification does not inherently protect privacy, but makes it easier to do so procedurally. It also improves reliability of the system, relaxing printing and storage requirements that would be necessary to achieve the same level of privacy in the original system. .

Chapter 6 ends the document with discussion of open problems and final thoughts. Appendix A lists directions for how to create a custom Linux LiveCD using the Ubuntu linux distribution.

Chapter 2

BACKGROUND

Voting systems are typically large, complex systems and certain technical design decisions are often mired in political issues. These characteristics make them hard to understand and properly conceptualize. In this chapter, we put voting systems into context by discussing their goals, some of the cryptographic tools available to build them, and some of the systems available.

2.1 Voting System Goals

The goal behind any "good" voting system is to produce an accurate result that inspires the confidence of voters. Producing an accurate results sounds like common sense, but this goal is vastly oversimplified – what exactly does "accurately" mean, and how is that accomplished? Others have tried to put this into context with witty anecdotes:

> The purpose of voting is not to convince the winner that he or she has won, but to convince the losers that they have lost. [1]

However, just because the losers are convinced they lost does not exempt them or their supporters from claiming fraud and questioning the legitimacy of the winner. Our

[1] The earliest writing of this anecdote, by Jeremy Bowers, can be found here: `http://www.jerf.org/iri/post/2441`

ideal voting system prevents false claims of foul play by providing significant proof if it exists. Such a system may not prevent a coup, but the system will not give it legitimacy that it would otherwise enjoy.

There is not yet a standard set of properties or goals that voting systems must meet because our ideal system is ambiguous, but we focused on integrity, coercion resistance, and verifiability in our development of Punchscan. These three goals stem from the philosophy that Punchscan should be as transparent and accurate as possible while maintaining the secret ballot. There are many other possible voting goals, these goals include: usability, accessibility, ease of administration, cost, scalability, reliability, recoverability. The following sections talk about each of these goals.

2.1.1 Integrity

Integrity in a voting system means that votes are counted as intended and it can be split into three different sub-properties: *cast as intended*, *recorded as cast*, and *tallied as recorded*. This split represents three different phases of the transfer of intention from the voter to the counting authority: a voter's expression of her intention, the recording of that intention to an official record that should be immutable, and the counting of that record. The first detailed mention of this split can be found in [42].

Notice that the counting step and intention step are always necessary in a voting system, or we could not produce election results. We need to count, and in order to do that we must know the intention of each voter. Also, observe that dropping the intermediate step can cause problems. Suppose we count as we receive intention without a separate record for each voter. If we make mistakes there is no way to go back to a record to correct our count. Therefore, these three properties are necessary, but not sufficient, goals for any voting system. The focus should be on how well a system accomplishes these goals, and not that it simply achieves them.

Cast as Intended *Cast as intended* represents the ability of the voter to transfer her intentions properly into the system, and cast her vote as she intended it to be cast. This could be a direct recording by putting pen to paper, or by using a device like a touch screen system which purports to record your intention for you.

We limit this property to mean that a voter can properly transfer her intention, and not necessarily that she succeed in doing so in real world conditions. An example of a system in which *cast as intended* is not possible would not always list all of the candidates in each race.

While all systems in use can be said to achieve this property, how well they support it is a matter of usability. For example, examine the butterfly ballot shown in Figure 2.1. It was used in the 2000 presidential election in Florida, and is said to have cost Al Gore the U.S. presidency [19] due to voters inability to cast their votes as intended.

Recorded as Cast When a vote is *recorded as cast* a device or medium used by each voter has properly recorded her intention to an official record, and this record has made it to the counting authority. This property can encompass a conversion from analog to digital representation, if that conversion is not an aggregation (*i.e.* there is no counting).

A system that weakly supports this property will faithfully record a vote as cast by the voter and will provide a mechanism for delivering it to the counting authority. The caveat with this goal is that many systems are incapable of *verifiably* adhering to this property, which is discussed in Section 2.1.3.

Again, if judged on a sliding scale metric, there are many systems that would not satisfy this property very well. There are also numerous problems with automated counting of paper records where votes are not counted, and this would be a recorded as cast problem. Most notable is the "hanging chad" [31], as illustrated in Figure 2.2. In this situation, properly maintained equipment would have prevented the problems, but built up wastepaper in

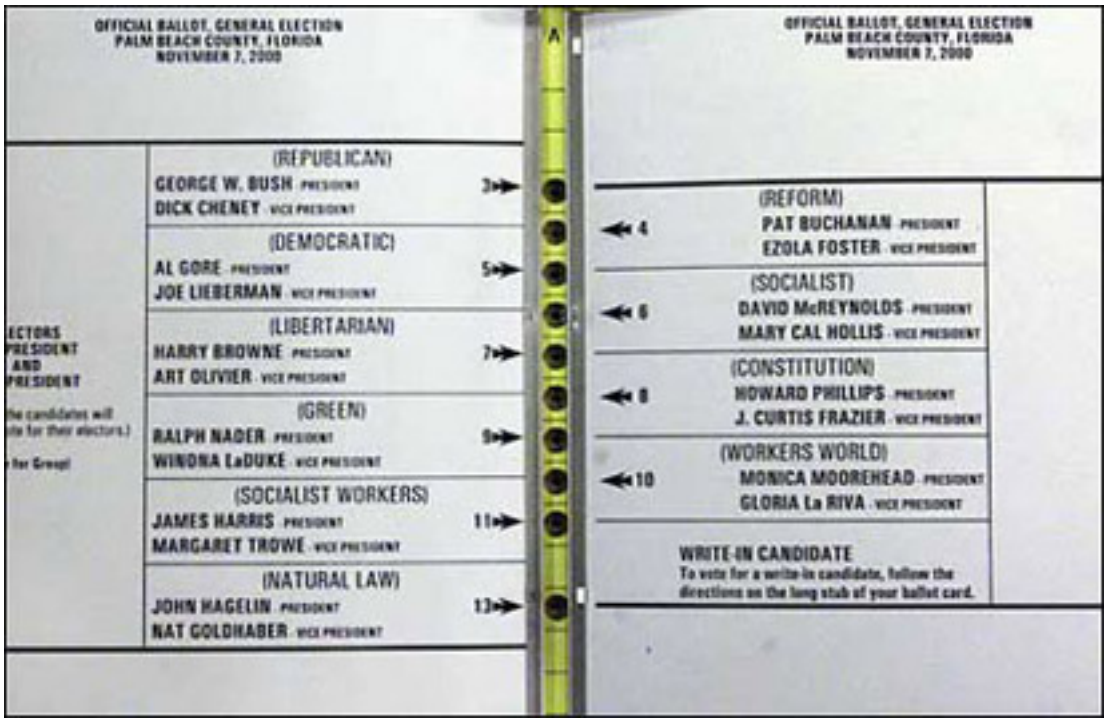

FIG. 2.1. **The Butterfly Ballot**. In the 2000 Palm Beach County Florida election, a voter who punched the second hole in the presidential race voted for Pat Buchanan rather than Al Gore, who was likely her intended choice.

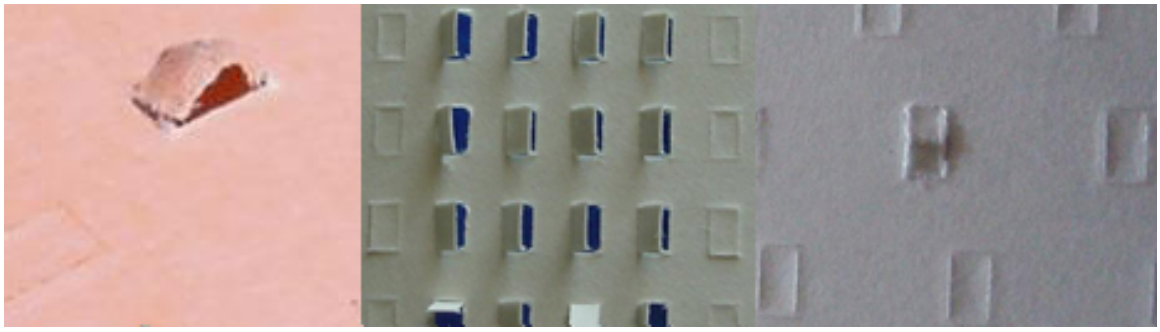

FIG. 2.2. **Hanging Chads**. From left to right: the pregnant, hanging, and dimpled chads that arise when punchcard machines are improperly maintained on election day. Only chads detached from the paper are counted by the punchcard machine.[3]

the machines and other failures caused by lack of maintenance caused them not to properly record each voter's intentions.

Counted as Recorded *Counted as recorded* means that from the official record, the counting authority is able to provide an accurate aggregation of the data. The counting authority can be a group of volunteers or machines, or a mix of both. The key properties are that the record is immutable and the counting is accurate. To accomplish these properties may require redundant counting by several imperfect entities to verify a proper total.

As is the case with the other two integrity goals, some systems may achieve this goal better than others. The biggest concern is how easily the record can be violated. This is a problem in most systems. Paper can be altered to invalidate votes, replaced, or destroyed and digital storage is, in general, easily manipulated.

Digital Recording Electronic (DRE) devices and their memory cards have consistently been shown not to support even basic protection mechanisms on their records properly [35, 46, 49, 28, 9, 23]. This problem is particularly problematic as they typically store only aggregate counts, and not full ballots for each voter. So, some counting is done on the machine, and totals from the machine are added with those from the other machines. The

a useful, attainable goal. At a minimum, we should expect to be reasonably protected from third party thugs that are not involved with the voting process, which appears to be what current systems try to accomplish. After that, the degree to which an insider must be involved to violate a voter's privacy and where we should set the threshhold is unclear because so much of a voting systems privacy depends on the environment in which it is used.

2.1.3 Verifiability

Verifiability is an added constraint on the properties we have discussed so far which many systems fail to achieve. For example, when we say that the *recorded as cast* property is verifiable, we mean that someone can check or audit to make sure that the property is working as the system is being used.

The question of who, how, and when is important when discussing the verifiability of any property. Recall that *cast as intended* means a voter is able to transfer her intentions to the system properly. This property should only be verifiable by the voter in the voting booth, because, if this property were verifiable at all times, the system could not be coercion resistant.

The proper recording and transmission of that vote to the counting authority is something that is also desireable to be verifiable by the voter, but this is where many systems do not succeed. Likewise, we would want everyone to be able to verify the counting process, but that is not typically available with current DRE and optical scan technology.

There is a careful relationship between verifiability in voting systems and coercion resistance. If certain properties or operations are verifiable, then we may lose coercion resistance. The goal should be to make things as verifiable as possible without losing coercion resistance.

2.1.4 Other Goals

Public systems are costly and meant to last for as long as possible, and if a failure occurs, there is a relatively ample time and opportunity to fix the problem or replace the system. Requirements for a voting system are unique in that it is a large scale system intended for use by the public for a short time that has no tolerance for failure and must be carried out with limited funds. An election is rerun only if there is catastrophic failure and a serious threat of open revolt. It needs a higher level of assurance than other systems.

With this in mind, we list other goals that voting systems should strive to maintain. These goals are not unique to voting systems, but the nature of a voting system makes these goals stand out:

1. **Useability and Accessibility.** The useability and accessibility requirements for a voting system are the most extreme of any system. Voters are not required to read a manual or be familiar with an interface, and they may have any number of disabilities. Directions at the polling place must be minimal. The system must be intuitive, and it should cater to voters with vision, hearing, and dexterity problems.

2. **Ease of Administration.** Elections are run by an army of volunteers. Often, the only requirement of a volunteer is that he or she should be able to read and write in the native language. Like accessibility, these requirements put the threshhold for ease of managing the system at an extreme level relative to other systems where professionals can be paid to operate and maintain the system. Administering the system should be obvious enough that volunteers are able to setup, operate, and close the polls with minimal technical knowledge.

3. **Cost and Durability.** Voting systems are typically used over the course of several days every other year, and it is hard to justify an expensive system. The system

should be as cheap as possible, and the equipment should be built to last as long as possible.

4. **Scalability.** The system should scale in several different ways. The first of which is that it should scale with the number of voters. The number of voters supported should grow as more equipment is added, and the counting processes should still finish in a reasonable amount of time after the election is complete. This timeliness requirement should also be true for the number of candidates per race and number of races per ballot. Last, the system should not have technical limitations preventing it from implementing election rules.

5. **Reliability.** It must be more reliable and resistant to disruption than the power grid, and any voting system must support a way to continue operation without power. Barring natural disaster or violent military actions that keep voters away from the polls, the system must work.

6. **Recoverability.** There should be ways to recover as best as is possible from catastrophic failures. In many cases this is a matter of redundancy and procedures (e.g. constantly publish the current count of votes each time period). The difficulty here is that you wish to achieve recoverability without drastically affecting the privacy, the rest of the system, or putting a lot of responsibility on volunteers.

There may be goals a voting system should meet that are not listed here, but these represent the requirements that are unique in a voting system. While public systems may nominally share many goals with voting systems, voting systems are unique in that they should have integrity, coercion resistance, and verifiability.

2.2 Building Blocks of Secure Voting

Below, we outline the types of existing technologies and ideas that are useful for a voting environment. These comprise a diverse set of tools with various security properties. Punchscan does not use all of these ideas, but we discuss them for completeness.

2.2.1 Basic Cryptographic Tools

Secure communication and authentication are often important in voting system protocols. The following basic cryptographic tools are essential:

1. **Encryption.** Encryption uses a secret to encode a message so that only the intended recipient with corresponding secret can read the message. It can be symmetric or asymmetric. In symmetric encryption, both the sender and recipient of the message share the same secret value. In an asymmetric encryption system, the sender uses a public code to lock the data, and the recipient uses a secret code to unlock it. This is similar to the recipient giving the sender an opened padlock. The sender uses the padlock to lock the message in a box, and the recipient unlocks it with the key.

2. **Hashing.** A cryptographic hash function takes a message of any length and produces an output message of fixed length. The output message from a hash function is computationally infeasible to reproduce without the original message. Another message with the same hash is also computationally infeasible to reproduce, and may not be possible.

3. **Digital Signatures.** A digital signature uses encryption and hashing to prove the identity of the sender of a message.

Many of the technologies described later in this section will make use of the properties these cryptographic tools provide. For further explanation of these cryptographic tools, we

direct the reader to [53].

2.2.2 Homomorphic Encryption

Homomorphic encryption is a cryptographic system which permits the ciphertexts to be added together to produce a summation that, when decrypted, produces a sum of the plaintexts. Suppose we have an encryption function E, decryption function D, key k, and numerical plaintexts p_1, p_2, then:

$$D_k(E_k(p_1 + p_2)) = D_k(E_k(p_1) + E_k(p_2)) \tag{2.1}$$

For our purposes, it is not necessary that the ciphertext be the same for both encryptions, only that the decryptions always produce the same result. Also, homomorphic systems are typically Public-Key systems [39], which provides an advantage as generating the ciphertext can be done by a third party without giving them information that might decrypt other ciphertexts. Examples of cryptosystems with this property are Paillier [44] and ElGamal [21].

2.2.3 Mixing

Introduced by Chaum [13], a mixnet decrypts a set of messages such that no one individual is able to determine the sources of the messages. This is analogous to voting in which paper slips are put into a box and it is shaken, effectively mixing the ballots so that

mixnet omprisets a sriges of which parically decryptput messages and

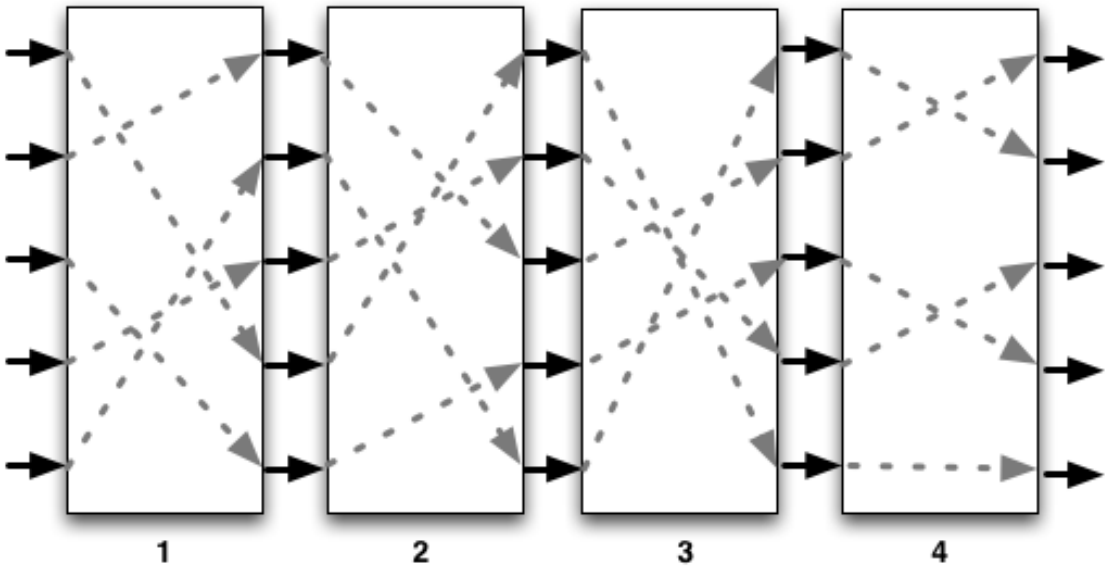

FIG. 2.3. **Mixnet**. Each of the four nodes in this example chaumian mixnet partially decrypt input messages, applies a random permutation to the order of the messages, and sends them to the next node in the mixnet. Messages are created for input to the mixnet by successively encrypting the message with the public key of each node in reverse order to hide the source of each message from an observer.

to the next node in the mixnet. To create the messages, the sender uses the public key, p, of each node and successively encrypts the message in reverse order. For example, with a mixnet of four nodes as in Figure 2.3 above, the mixnet encryption, ME, of each message, m, would be:

$$ME(m) = E_{p_1}(E_{p_2}(E_{p_3}(E_{p_4}(m)))) \tag{2.2}$$

The mixnet performs the decryption operation with the private key, k, of each node:

$$m = D_{k_1}(D_{k_2}(D_{k_3}(D_{k_4}(ME(m))))) \tag{2.3}$$

What we have shown so far is known as a *chaumian* or *decryption* mixnet, and any asymmetric encryption algorithm can be used in a decryption mixnet. The minimum requirement of an algorithm to create a mixnet are that the inputs be untraceable to (or different from) the output at each node. For example, *reencryption* mixnets differ from *chaumian* mixnets by partially decrypting and reencrypting each message for the next node. These mixnets can tolerate the failure of a threshhold of nodes, and each node will produce a unique output even with the same input.

To make a mixnet robust, or verifiable, a technique such as *Randomized Partial Checking* (RPC) [30] can be used. In RPC and other techniques, parts, but not all, of the mixing and decryption of the mixnet are published after they were performed such that an observer can have a high confidence that messages were not changed during the mixing process. In Figure 2.4 shows RPC in action.

Note that verifying a mixnet through RPC or another method can reduce the untraceability that the mixnet provides. For example, in the mixnet without RPC, only one node needed to keep its operations secret in order to maintain privacy. In the RPC version, two nodes are needed. We can fix this situation by having each node perform two mixes.

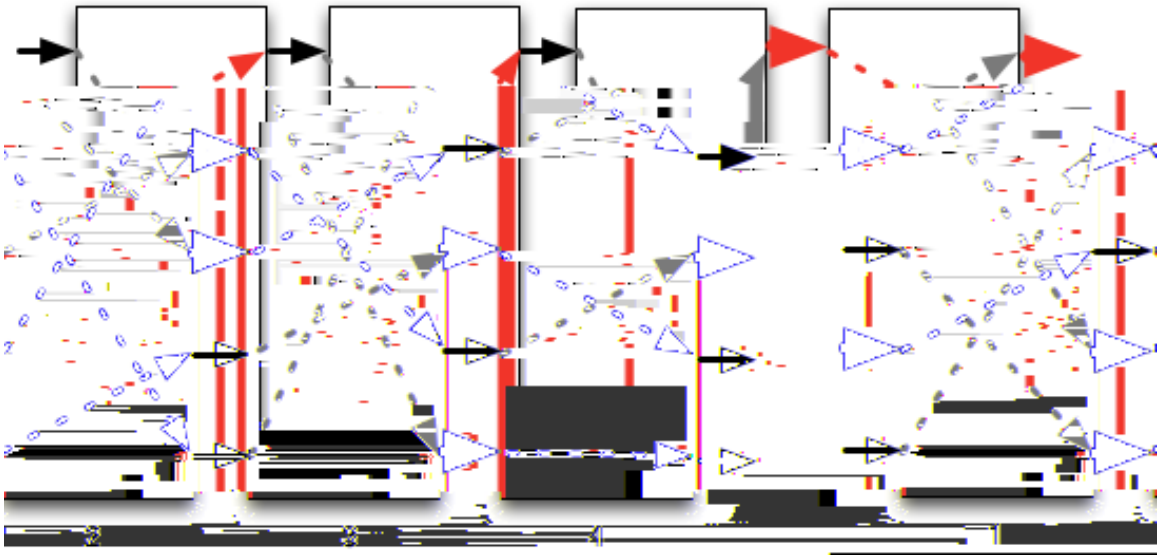

FIG. 2.4. **Mixnet with Randomized Partial Checking (RPC)**. Mixnets with RPC operate in the same manner as a normal mixnet. However, after mixing is done it reveals part of the data. The second and fourth messages in this illustration are checked or audited, and this choice cascades through the mixnet, as the three other messages not chosen for the first node are chosen to be audited for the next node.

2.2.4 Blind Signatures

Also introduced by Chaum [14], blind signatures allow a signing authority to sign an encrypted message without knowing the contents of the message. The decrypted message can later be publicly verified with the signature. This is analogous to putting a message and a piece of carbon paper inside an envelope, and having the signing authority sign the envelope. We can check that it was signed by the signing authority both before and after we open the envelope (due to the carbon paper). These signatures have been used in voting protocols where voters generate their ballots with a voting server's public key and have the signing authority sign them with the private key of a separate registration server before anonymously submitting them to the counting authority [26].

2.2.5 Commitment

A commitment [10] allows a message to be released later, but prevents the sender from modifying the message after send the commitment. Such schemes can be unconditionally secure or unconditionally secret. An unconditionally secure bit commitment means that the message is impossible to change, and that it is computationally infeasible to determine the contents of the message without the original message. By contrast, the unconditionally secret commitment makes the message computationally infeasible to change, but impossible to determine, with certainty, the content of the message. Commitment has been used in verifiable secret sharing [18], verifiable mixing [45], and cut and choose proofs [16].

2.2.6 Secret Sharing

Secret sharing schemes (SSS) are schemes in which one individual, the dealer, shares a secret with a group of participants such that certain sets of participants can recover the secret while other sets cannot. In a perfect SSS, any combination of the sets of participants who are unable to recover the secret cannot ascertain any more information about the secret than an outsider.

Threshold versions of these schemes—where all that is needed to recover the secret is any k out of n of the group of participants—were first introduced independently by Blakley [8] and Shamir [50] in 1979. Since then there has been a significant amount of research in the field. We briefly explain Blakley and Shamir's schemes below.

Shamir Secret Sharing Shamir's scheme [50] is based on the fact that k coordinates on a graph (x, y) will uniquely define a polynomial of the degree $k - 1$. That is, two points define a line, three define a parabola, four define a cubic curve, and so on. Once a secret is generated by the dealer, he can generate a curve that intersects on the y-axis of the graph with n coordinates on the curve to give to each participant. Any k of those participants are

then able to regenerate the curve using polynomial interpolation in order to find the point at which it intercepts the y-axis, the dealer's secret. In order to avoid dealing with complex fractions the curves generated by the dealer are over a finite field.

Blakley Secret Sharing In Blakley's scheme [8], the dealer picks a point and creates a set of n n-dimensional hyperplanes whose intersection contains that point. The dealer then distributes the information to define each hyperplane to the participants and a specific coordinate to look at after the group determines the intersection points of the plane.

While it does not look like it, this scheme is similar to Shamir's scheme when you realize that two non-parallel lines intersect at a point, three non-parallel planes intersect at a point, and so on. Note that you cannot encode a secret using all coordinates, because a participant would know if the secret were located on his plane. Shamir's scheme is more space efficient than Blakley's, because Blakley's requires that each share be k times larger than the secret, whereas Shamir's scheme has shares that are the same size as the secret (all x coordinates in Shamir's scheme can be known to all participants).

Verifiable or Anti-Cheater Secret Sharing In some SSS, users may introduce incorrect shares to prevent proper decryption of the key without detection and possibly other unknown reasons. It works by adding additional information that allows participants to verify their shares are consistent. This is typically in the form of public parameters or values, which are used to compute something with the share that each participant makes public that should match a given value once added all together. Chor *et al.* [18] were first to propose such systems.

2.2.7 Cut and Choose Protocols

The most important tool used in Punchscan is the cryptographic principle of cut and choose (CNC) [16]. A classic CNC protocol involves two participants: Alice who makes a claim about a piece of data, and Bob who will attempt to verify that claim without seeing the data. To verify Alice's claim, Bob asks Alice to create many substitutable data blocks and to encrypt them. Bob then *chooses* the data block to use, and Alice gives Bob the decryption keys for the remaining *cut* data blocks to check that they meet Alice's claim. For example, Alice may claim that a sealed envelope contains a hundred dollar bill but she does not want Bob to see the serial number on it. Bob can ask Alice to produce n sealed envelopes, each with a hundred dollar bill in them. Bob opens all but one of the envelopes and if they all satisfy Alice's claim, he is reasonably sure that the chosen one does as well without opening it. In order to cheat successfully without being caught, Alice creates one bad data block, and hopes, with probability $1/n$ that Bob will choose it. If the number of blocks, n, is large enough, there is an overwhelming probability that Alice will get caught if she tries to cheat.

When we use CNC in voting, Bob is an auditor of election data. The reason Bob cannot know the contents of the data block is to protect voter privacy, but we wish simultaneously to permit a transparent audit that can be checked by the public. By allowing Bob to choose some of the data to audit, we can achieve both goals. A notable difference in Punchscan from the types of CNC protocols described above is that our CNC constructions give Alice a 50% chance to cheat but she must make that choice many times over. So instead of succeeding with probability $1/n$, she succeeds with $1/2^n$. As n grows, this quickly results in an even greater probability of being caught, even when she merely tries to change one vote.

2.2.8 Software Attestation or Validation

We define software validation to be the process by which a user ensures that they are running the software they intend to run. As it is unreasonable to expect a user to evaluate source code and generate a binary executable, this definition assumes that there is some trusted third party (in our case, an independent testing authority (ITA)) that will perform these functions, and that everything is publicly posted so that any willing individual can check that the binary was well-formed.

This problem is similar to the problem of software distribution over the internet discussed by Rubin [47]. He proposed Bellcore's Trusted Software Integrity System (Betsi), in which authors could uniquely register themselves, and any software they wish to distribute could be issued a certificate containing, among other information, a cryptographic hash of the software. Even if the author's site were compromised users could be sure they were running the author's software by checking the certificate and the hash of the file they downloaded.

Unfortunately a system like this never caught on. But, as can easily be seen today, use of cryptographic hash functions has become a common practice when performing software installation [3, 7, 4]. This practice is not much better than an integrity check when the software is downloaded from the author's website because an attacker changing the software could easily change the hash. However, there are now a significant number of mirrors and download sites which host many applications, and many authors host their software only on these sites. With the existence of these much larger targets, an author's site is unlikely to be compromised, so retrieving the hash from the authors site to check downloaded software gives the user a reasonable level of assurance.

Cryptographic hash functions have also been used in systems like Tripwire [34], which detect unauthorized changes to sensitive files in an operating system. In this case they are

very effective in determining if tampering has taken place, since the user has a record of the hash values of each file.

2.3 Related Work

The American Election Assistance Commission (EAC) has released several sets of voluntary voting system guidelines [20]. These guidelines include a description of "End to End Cryptographic Independent Verification" (E2E) systems. According to the most current guidelines released by the EAC in 2005, E2E voting systems have the following properties:

1. They record voters ballot selections at electronic voting machines and encrypt the records of votes for later counting by designated trustees.

2. They produce a receipt that can be used by the voter in a process defined by voting officials that would enable the voter to verify that the voter's ballot selections were recorded correctly and counted in the election. The receipt preserves voter privacy by not containing any information that can be used to show the voter's selections.

3. No one designated trustee is able to decrypt the records; decryption of the records is performed by a process that involves multiple designated trustees.

4. The process used to verify that ballot selections were recorded correctly and counted preserves voter privacy by not revealing any information that can be used to identify the voter's selections.

5. End to end systems store backup records of voter ballot selections that can be used in contingencies such as damage or loss of its counted records.

6. The backup records contain unique identifiers that correspond to unique identifiers in its counted records, and the backup records are digitally signed so that they can be verified for their authenticity and integrity in audits.

As it is paper-based, Punchscan meets all but the first property as described by the EAC. It is among a number of other systems with similar properties. We describe the more recent and well-known E2E systems in the following sections.

2.3.1 MarkPledge

VoteHere's MarkPledge was developed by Neff [40, 52]. It uses a DRE or a device that connects to a DRE to produce an encrypted ballot receipt that can be decrypted through a mixnet. MarkPledge provides the only voter-verifiable receipt that does not rely on information being destroyed by each voter before it is publicly posted, but requires a specific protocol that must by followed by the voter and involved matching short strings of characters.

The MarkPledge protocol can be described as follows:

1. Voter enters a voting booth, a privacy preserving environment, and uses a token that activates the machine for voting.

2. Voter tells the machine her choice.

3. Machine creates an encrypted ballot representing the voter's choice and sends a unique ballot serial number, and the encrypted ballot to the printer.

4. Voter tells the machine a random string that acts as a *challenge* to reveal part of the encrypted ballot.

5. Machine prints the *challenge* and the response to the commitment, which is also filled with dummy data to protect voter privacy.

6. If the voter OK's the choice, the machine posts the encrypted ballot and the voter can take the printed receipt.

The encrypted ballot and the challenge are specific to MarkPledge. For each candidate on the ballot, a row of pairs of 1-bit strings is created. If the voter chose that candidate, each pair is $1, 1$ or $0, 0$. If the candidate was not chosen, each pair is either $1, 0$ or $0, 1$. Also included on the encrypted ballot is a row of pledge bits. Each bit is 0 or 1, corresponding to the pair in the chosen row.

When a voter makes her challenge, the left or right half of each chosen pair is revealed, and the voter can later check to make sure the receipt was well formed by using the public key to recompute and compare the encrypted ciphertext on the receipt. Anonymity is preserved because the machine will also open half of the rest of the pairs corresponding to the pledges.

The openings and challenges, from the voter's perspective, can be represented by strings of characters. This representation makes the system more practical for a voter to use. To verify the receipt, a voter must use a trusted computer that can perform the encryption options. Variations of the encrypted ballot can support ranked voting and other methods.

2.3.2 SureVote

SureVote is David Chaum's first proposed E2E voting system [11, 15, 12] that uses visual cryptography [38]. It uses a DRE with special printer to print an overlay of two transparent sheets and see a printout of their choice. By destroying one of the sheets, a voter can encrypt her choice because neither sheet has enough information to reconstruct the original message.

Figure 2.5 shows an illustration of visual cryptography. SureVote prints out both

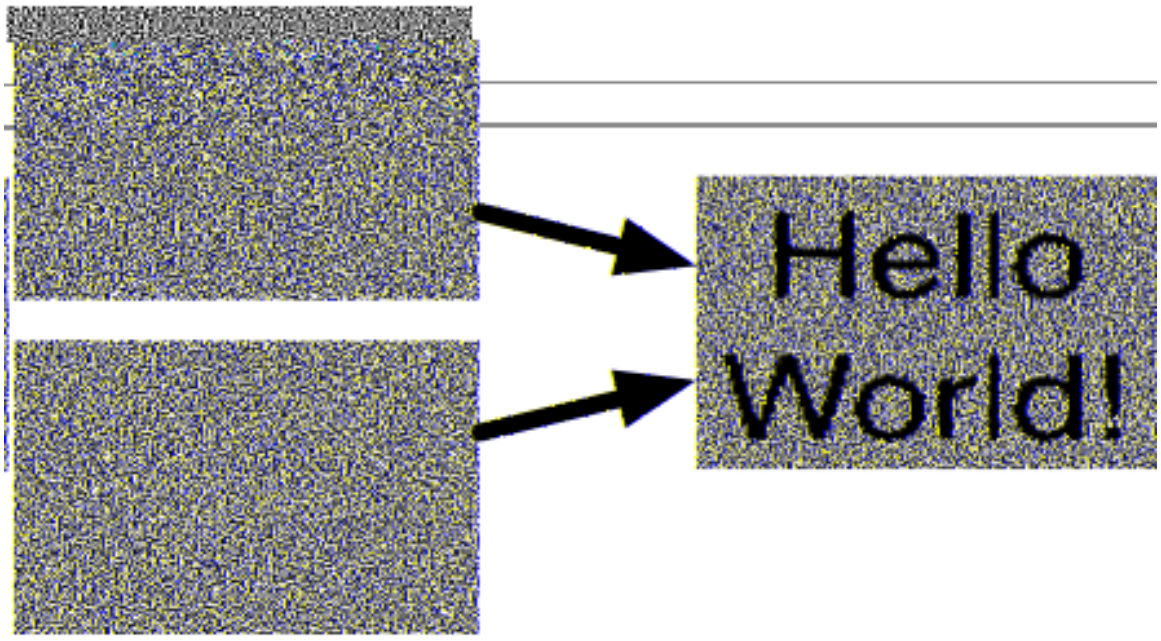

FIG. 2.5. **Visual Cryptography.** An example of visual cryptography. Pixels are grouped to produce partially white and black spaces when the two sheets are overlayed. When the sheets are separated, both sheets appear to be partially white to the human eye. Destroying one sheet makes it impossible to determine what was shown on both sheets without reconstructing the destroyed sheet.[5]

sheets overlayed, and the voter must separate and destroy one of she sheets, keeping the surviving receipt as a receipt. The surviving sheet and separate encryptions for the sheets are publicly posted. SureVote uses a mixnet to reconstruct the content of the destroyed sheet. Proper construction of the sheets are verified by opening the encryption of the receipt sheet.

The chief disadvantages of SureVote and MarkPledge are that they require more random bits to represent the choice of the voter than are needed. This makes them vulnerable to subliminal channel attacks which could secretly reveal voter choices on a large scale as shown by Karlof et al. [32]. Karlof et al. also argued that the protocol voters must follow in MarkPledge may confuse voters, and showed that voter's may not notice subtle changes in the protocol that would compromise integrity of the system.

2.3.3 Prêt à Voter

In 2005, Ryan and Chaum proposed Prêt à Voter as an improvement of SureVote using a preprinted ballot [17]. In it, candidate names are printed in a vertical random permutation and an encryption of the candidate list, called the *onion*, is printed at the bottom of the ballot. A voter votes by marking next to the desired candidate on the list and destroying the candidate list (the left half). The surviving part of the ballot with marked positions (the right half) is scanned and taken home by the voter to be checked online.

The position and the *onion* are used in a mixnet to decrypt the Prêt à Voter ballot. The onion is small, and could also be a hash of the encrypted list, limiting the ability to embed a subliminal channel into the printed ballot. Some of the ballots created the system must be spoiled and audited in order to check that the ballots are being printed correctly.

Punchscan is a close relative of Prêt à Voter but has several key differences. Punchscan does not use a mixnet. The structure created is very similar to one but allows for much faster computation of results. The Punchscan ballot does not have an *onion*, and uses a different format from Prêt à Voter.

Chapter 3

OVERVIEW OF THE PUNCHSCAN VOTING SYSTEM

Punchscan is an end-toend (E2E) cryptographic voting system that provides a high level of transparency throughout the entire election process. It uses privacy-preserving receipts to create an immutable public record. A mandatory auditing of encrypted data ensures with an overwhelmingly high statistical degree of confidence that this public record is decrypted properly by elections officials.

We begin the overview of Punchscan by explaining the voter experience, followed by an explanation of the system architecture in Section 3.2. Section 3.4 explains the specifics of the counting and auditing mechanisms in Punchscan. Portions of these sections are derived with permission from the co-authors in [25, 45]. The security problems that we are addressing in Punchscan are explained in Section 3.6.

3.1 Voter Experience

The Punchscan ballot, as illustrated in Figure 3.1 is created by combining a top and bottom sheet of paper. The top sheet has letters or symbols next to candidate names and holes in it to show letters that are printed on the bottom sheet. The letters on both sheets are ordered randomly.

To vote, each voter uses a bingo dauber to mark the letter seen on the bottom sheet

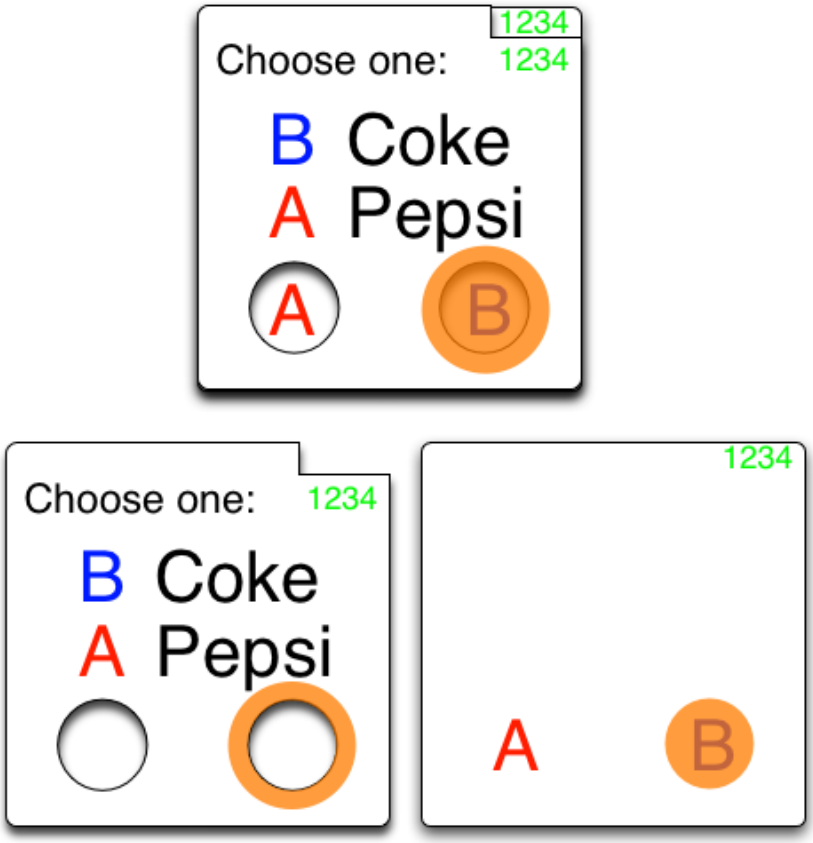

FIG. 3.1. **The Punchscan Ballot**. On top, a marked ballot. On the top corner the serial number is listed twice, once on each sheet. In the center, the ordered candidate list, and to the left a random ordering of symbols (A or B in this case). Below the candidate list, printed on the bottom sheet, is another independent ordering of the same symbols. Underneath we see that both sheets are marked after the top and bottom sheets have been separated. A top or bottom sheet by itself does not reveal any useful information about how the voter has voted.

that is next to the candidate of her choice on the top sheet. This action creates a mark on both sheets, because the bingo dauber is larger than the hole through which the letter is viewed. Afterward, the voter destroys either the top or the bottom sheet, and the surviving sheet is scanned, publicly posted, and kept by the voter as a receipt.[1] The choice of which sheet is destroyed is determined at random before the voter views the ballot. This choice can be made in a variety of ways, for example, we could have a jar with an equal number of tokens with "top" and "bottom" written on them, that are then chosen by or in front of the voter.

As shown in Figure 3.1, neither half of the ballot can reveal the original vote by itself. Only the Trustees can determine the original intent, and it does so using the Punchboard explained in Section 3.5.1.

3.2 System Architecture

The Punchscan Architecture is illustrated in Figure 3.2. A web server acts as the central repository for all election data. The Trustees use a trusted workstation to create the ballot images, generate the election data, and create commitments to the generated election data that are posted on the webserver. Later, the election data is challenged by an Auditor, and the trustees must post the information necessary to reveal the commitment data. Anyone can verify the proper disclosure of the challenged data.

Trustees also use a printer to print the Punchscan ballot images and send them to polling places to give to each Voter on election day. Each voter receives a privacy preserving ballot receipt after voting, and is able to verify that the receipt is posted on the web

[1] Alternatively, the top sheet of the ballot could consist of a random ordering of the candidates, and the voter would mark next to the candidate as is done in PAV. This change removes the need for a choice of sheet (forcing the bottom sheet to be the choice). This modification is discouraged for two reasons: we wish to preserve an ordered candidate list as may be required by state laws, and the random choice serves as an automatic CNC audit, as it is defined in Section 2.2.7, to check the integrity of the printing process.

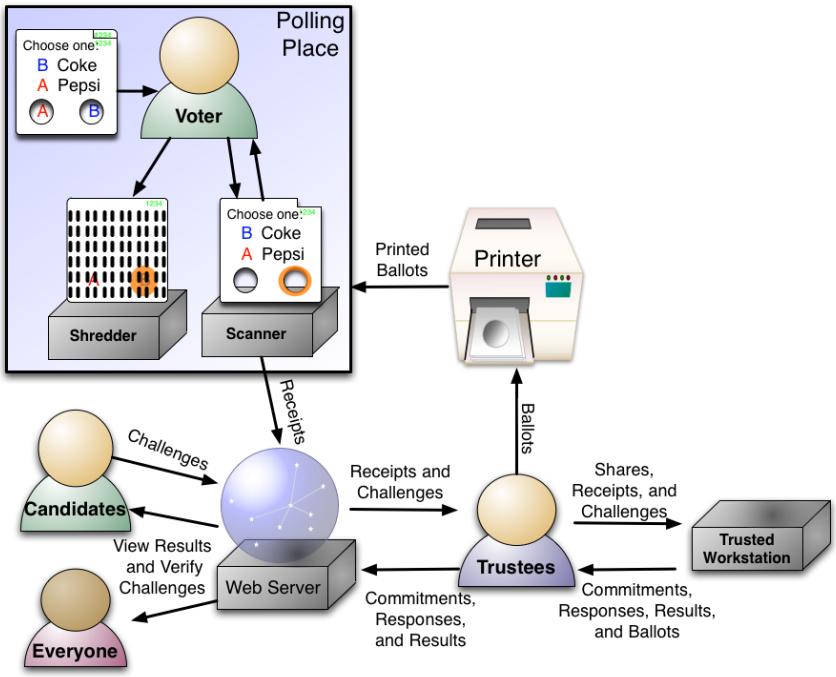

FIG. 3.2. **Punchscan Architecture**. An architecture diagram of the Punchscan System. The web server acts as a central, transparent repository for all election data. Everyone can check that the data is correct.

server. Everyone can check a ballot, view the results, and verify the audit data posted on the webserver.

3.2.1 Web Server

A Web Server or group of Web Servers serves as the communications hub for all election parties. It is used to post receipts and Punchboard data. Auditors also use the Web Server to submit challenge and audit requests. The Trustees respond to these requests by updating the copy of the Punchboard stored on the server(s). While this server contains important election data, its corruption via hardware failure or malicious attack does not imply voter privacy or election integrity has been violated. All data can be regenerated or uploaded from backups at any point by Election Authorities, and the election protocol can continue when the Web Server is reestablished. However, since news that the Web Server has been compromised might adversely affect voter confidence, it is still important to keep the server properly secured and maintained.

As the central communications hub for election participants, the Web Server performs many important functions. When voters enter the Ballot ID from their receipt, the server's Web Application Software accesses mark and ballot configuration data from the public Punchboard to render a virtual copy of the receipt. Voters can inspect this virtual copy to ensure it is identical to their paper receipt. Observers can download all public election data, including the Punchboard, from the server in an open data format for automated processing or manual inspection. At the appropriate times, the server will accept challenges and audit requests from authenticated election Auditors. In response, Election Officials must be able to log onto the Web Server to upload updated election data securely. Only Auditors and Election Officials require authenticated access to the server; all other users may remain anonymous. All data and software on the Web Server are public, therefore there is no risk a malicious user could obtain sensitive data.

3.2.2 Trusted/Diskless Workstation

While the Web Server is a public and marginally expendable computer, Election Authorities require a special, high-security Trusted Workstation with which they can process important election data with verified software. The workstation has no need of a hard drive and therefore should contain no information or programs when it is not in use. The Workstation also has no network interface or modem. Election Officials supply an operating system, programs and election data on removable media that is posted online for anyone to check before or after an election, and program output is stored on recordable media before the workstation is powered down. The Punchboard ensures the integrity of election, so the reason for the high-security of the workstation is to protect voter privacy. This component is discussed in detail in Chapter 4.

3.2.3 Printer

Since Punchscan is a hybrid paper/electronic voting system, separate hardware is necessary to manage and process paper ballots and receipts. Paper ballots can be printed with an ordinary inkjet Printer, although for large elections this task may be delegated to an industrial printing firm. The Printer must be audited to ensure that each ballot is printed correcty. The printer is trusted with keeping the information printed on the ballots secret. Printing distribution strategies can be used to minimize the impact of a violation of the trust placed in the printer.

3.2.4 Scanner

Within the polling place, each voter marks her ballot and separates its pages. One page is destroyed by a cross-cut paper Shredder with a battery backup. Shredded ballot pages are properly disposed of using standard procedures for handling sensitive documents.

The remaining page is scanned using an optical Scanner with battery backup attached to a computer workstation. The workstation includes software to detect marks made by the Voter and a screen to allow for corrections and final confirmation. Once verified, the vote is encoded in a digital format as a list of marks on a specified ballot page. The file is transmitted to the Web Server or stored on removable storage for later hand delivery. The Scanner must be properly calibrated to recognize all possible valid marks on each ballot. This calibration can be done using software algorithms or by calibrating the Scanner with a sample ballot with all positions marked.

3.2.5 Ballot Authoring

One final software program is needed to specify key ballot parameters. The Trustees use any program to author the ballots, with special graphical elements that are recognized by an automated application. The program outputs a standard file containing this information, which is transmitted to the Web Server for public examination. Once all errors have been detected and corrected, the file is locked to prevent further editing.

3.3 User Roles

There are a few defined roles that users play in the system, but anyone, if they choose to do so, can play the role of observer by looking at publicly provided data and verifying their correctness. We now present and describe the four roles of Election Authority, Poll Worker, Voter, and Auditor.

3.3.1 Election Authority

including the Punchboard, in both its encrypted and unencrypted forms. It is essential that the election machinery distribute access to sensitive data across a majority of trustees (e.g. 5 of 7), such no that trustee could ever view the unencrypted secret election data by himself.

3.3.2 Poll Worker

A poll worker is the volunteer or other election official who is responsible for the proper operation of each polling place. Poll Workers perform various roles such as manning tables, and passing out ballots.

3.3.3 Voter

Voters contribute to any election system by casting ballots. Because Punchscan is an E2E system voters may also play an additional role as independent auditors. They are encouraged to use a website to verify that the receipt they hold in their hand matches what was counted in the tally.

3.3.4 Auditor

Auditors form the basis of the independent verification, and perform audit checks on Punchboard data. Auditors along with any interested observers can examine E2E data to verify the election proceeded without irregularities or tampering. Obviously, it is important that the Auditors remain independent from the Trustees since collusion between these groups could violate election integrity and voter privacy. Any observer can view audit data and verify its correctness; auditors are different because they are tasked with making the random choices required by the system.

3.4 Core Components

At its core, Punchscan is a derivative of SureVote, an earlier E2E system (as described in section 2.3) proposed by Chaum that used visual cryptography [15]. Like SureVote, Punchscan uses a two-sheet ballot and a cryptographically assured decryption process, but both of these components are redesigned to be substantially less complex in Punchscan.

The key advantage of Punchscan over SureVote is that the voter marks a pre-printed ballot instead of trusting a machine to generate one. The ballot is still split by the voter but Punchscan does not use visual cryptography. The new ballot is an improvement because the voter verifies that her positions and letters are correct and need not make an exact comparison of the position of black pixels on a screen. In a sense, the Punchscan ballot receipt is human readable even though it does not reveal information about choices on the receipt. It can also be used to provide a receipt for mail-in ballots.

Instead of a series of tellers, as defined in Section 2.2.3, performing mix operations, Punchscan relies on a simple auditable system with a function similar to a mixnet using two cryptographic primitives: a cryptographically secure pseudo random number generator (CSPRNG) [37] and an unconditionally secure bit commitment scheme (USBCS) [33]. There are some trade-offs in losing the tellers, but the mix still permits us to achieve the security goals of *unconditional integrity* and *computational privacy* as described in Section 2.2.5.

Now, we discuss the ballot, our mixnet-like system, the Punchboard, and give an overview of the auditing process to verify that the inputs to the mix net are an accurate representation of the votes cast in a Punchscan election.

3.5 Digitizing the Ballot

The position marked by the voter is known as the *mark position*, and in subsequent diagrams is either 0 for the left mark or 1 for the right mark. For races with more than two candidates, we would indicate the choice as 0, 1, ..., or n, with numbering starting at the leftmost position.

In implementation, the ballot could be represented as a full permutation or by a cyclic shift. The cyclic shift uses modulo arithmetic, and the vote position can be added to the two cyclic shifts in order to determine who is chosen. In the simple case, one bit may be used for both candidate orderings on each ballot sheet (0 for AB and 1 for BA). Representing arbitrary permutations requires more information. As shown in Figure 3.3, we can create four combinations of different ballot types.

In cyclic shifting when adding (mod 2) the sheets together, adding the mark position produces the final vote. More specifically, if both the top and bottom are 0 or 1 ($0 + 0 = 0$ and $1+1 = 0$), then the 0 position chooses the first candidate, and 1 chooses the second, and if the top and bottom differ ($1+0 = 1$ and $0+1 = 0$), then 0 chooses the second candidate, and 1 chooses the first (just as it is seen by the voter in Figure 3.1). This generalizes to N candidates with mod N arithmetic.

In the cyclic case the Punchboard merely takes as input the position and performs cyclic operations to turn the ballot back into canonical form (where the shift turns to 0), and the number left over was the choice made by the voter. A consequence of cyclic shifting is that it can reveal information in elections where multiple choices are being made, such as in m out of n elections or ranking methods like instant runoff voting, because the spacing between the candidate lists ane the choices would indicate certain unique voting choices.

If we use a permutation, the Punchboard performs operations that reverse the ballot permutation, and the position is changed during this process. In the current implementation

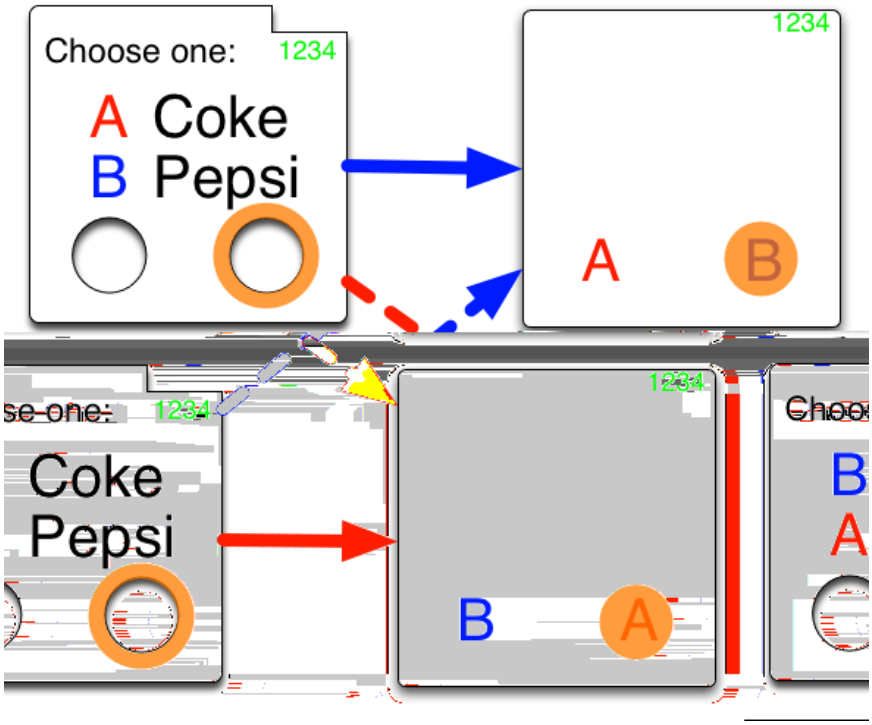

FIG. 3.3. **Possible Ballot Combinations**. All possible combinations of a two-candidate ballot. Notice that while there are 4 combinations, only 2 of them are unique when combined with a vote position.

FIG. 3.4. **The Unredacted Punchboard**. Coded votes are displayed on the left side in the Print (P) table and uncoded, canonical votes are in the Results (R) table. The Flip columns in the Decode (D) table contain either a straight arrow, which leaves the vote position mark alone, or a circular arrow which flips a 0 to 1 and a 1 to 0. The top and bottom sheet columns considered together should match the Flip 1 and Flip 2 columns considered together such that 0 corresponds to the first candidate in the R table and 1 the second.

it does not matter what sheet is taken as a receipt, as it is the combination of both receipts that is proved by the Punchboard, and not the individual sheets.

3.5.1 Punchboard

In order to determine voter intent, the Trustees must know the letter ordering on at least the destroyed half of the ballot, and this information is available through the Punchboard, shown in Figure 3.4. To interpret the results, candidate order is associated with a marked position in the Results (R) table Thus, a 0 position in the R table represents a vote for the first candidate listed on the ballot (Coke).

We use the Punchboard to provide voter privacy and election integrity. If we post it as shown in Figure 3.4, there is no privacy in the system, but if it remains secret, we provide no publicly verifiable integrity to the counting process. In order to achieve both of these properties, Punchscan uses its own USBCS to commit to certain data before ballots are printed for the election, and CNC is used to reveal parts as the election progresses. The Punchboard performs an operation, reveals an intermediate result, and then performs a second operation to reveal a final result. Either the intermediate operation or the final operation is revealed, and this method enforces integrity by making public certain values as we progress through the election after they have been committed too by the Trustees, allowing anyone interested to check to make sure the public values, or revealed data, match what election authorities committed to before the election. The data not made public protect the privacy of voters.

In Figure 3.5, the boxes covering the table cells represent committed data using the USBCS. While the commitment function is the same, the data put into it and the functions of certain commitments can be split into three types, each corresponding to a different CNC operation.

The first type of commitment, the printing commitment, is a commitment of each cell for the top and bottom sheets in the P table. The printing commitment data are revealed when a ballot is spoiled, or after results are posted when the Trustees knows which sheet each voter took as a receipt. Thus, depending on the sheet chosen to be destroyed, the receipt not only verifies the positions chosen by the voter but also permits voters to check on the printing process.

The second type of commitment is a D-row commitment, which encompasses both flips in the D table and the corresponding permutations. The data for a ballot is released only when it is spoiled. Revealing the data checks on both the printing process and serves as an integrity check to make sure the flip columns correspond to the top and bottom sheet

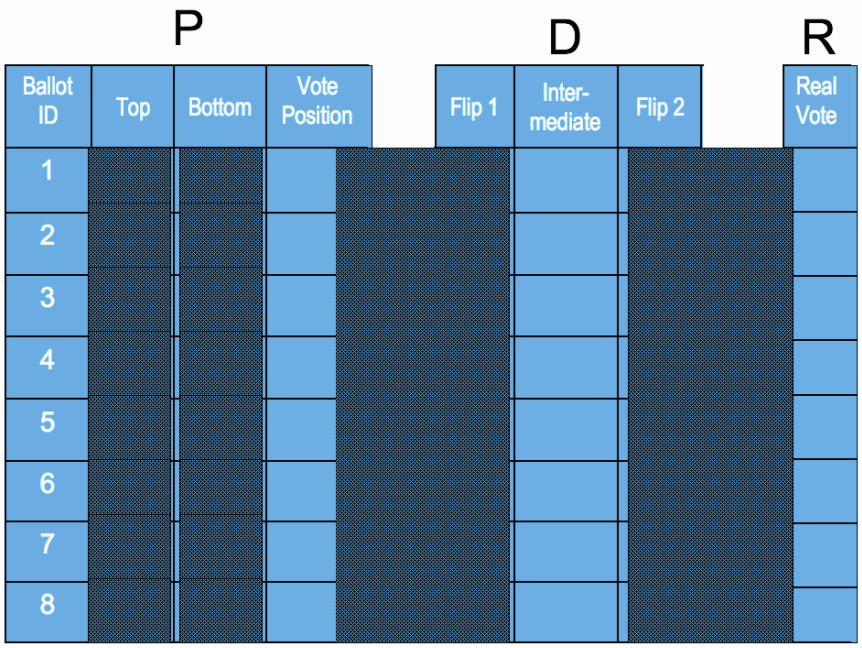

FIG. 3.5. **Pre-Election Punchboard**. The punchboard as published before any auditing. Each cell in the P table is committed to using a USBCS. Each Flip column in the D table, and the rows in the P and R tables it corresponds too are also committed.

columns in the table.

The last kind of commitment, the mix commitment, consists of each of the entire flip columns (i.e. Flip 1 and the permutation to the P table or Flip 2 and the permutation to the R table). After results are posted the auditor chooses which of the two commitments for each D table to reveal. This choice is a CNC operation, and allows anyone to verify that the table was filled out correctly by the Trustees, but prevents anyone from determining what rows or votes in the P table corresponded to what rows or votes in the R table.

3.5.2 Auditing

There are three types of audits: pre-election, results posting, and post-election. Because a malicious person does not know what data will be chosen by the auditors, any

malicious action taken has a high risk of being caught. Thus, it is important that the Trustees commit to the election data before the auditors perform any actions, because prior knowledge of intended auditor actions would let the Trustees or the attackers know what malicious actions they could take without being detected.

Pre-Election Audit. The pre-election audit ensures proper construction of the Punchboard. After the Trustees first publish the Punchboard, Auditors choose half of the rows in the P table at random and the Trustees publish the contents of those rows. The published rows are checked with their commitments to ensure that they are well-formed, they are then discarded and the remaining rows are used to print the sheets that make up each ballot used in the election. A sufficient number of the printed ballots should also be audited and spoiled in the same way to check on the printing process. This audit makes half of the rows unusable, so the Trustees must generate more than twice as many rows as the number of needed ballots.

Posting Results. When results are posted, the Trustees populate the Vote Position, Intermediate Position, and Real Vote columns of the tables. They additionally reveal the sheet that each voter took home as a receipt. Each voter is able to verify that her ballot was included with the correct marks in the final tally, that her receipt matches the revealed data, and that it was well-formed. Everyone is able to verify that revealed data matches what was committed to before the election. An illustration of the Punchboard after the results are posted is shown in Figure 3.6.

Post-Election Audit. The post-election audit ensures that the counting process was executed properly while maintaining voter privacy. For each published D table, auditors choose the two columns left or right of the Intermediate Position column and the Trustees reveals that data. That way, everyone can then check that the marked positions match the intermediary values, or that the intermediary values match the final results. Because the Trustees or attackers did not know what half of each D table would be selected before they

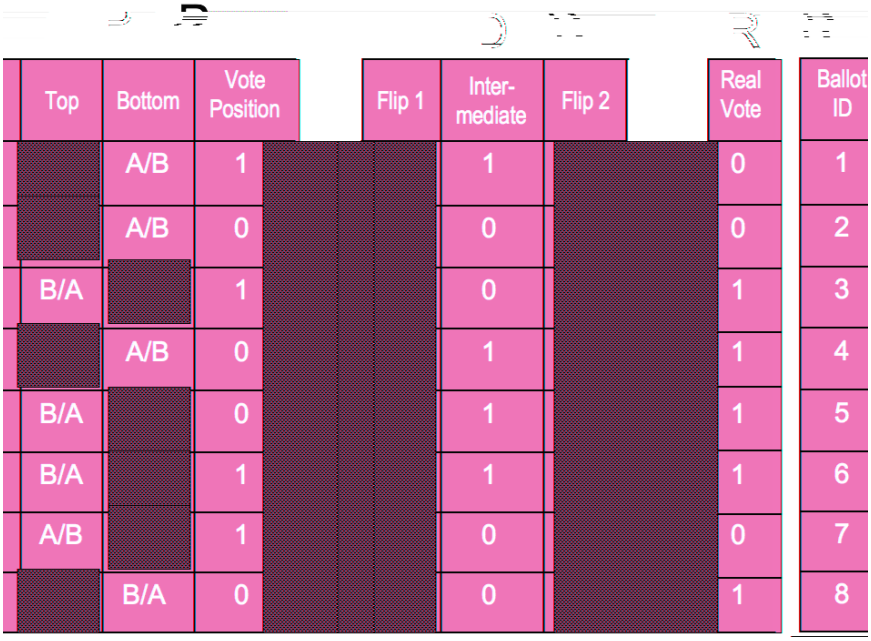

FIG. 3.6. **Post-Election Punchboard**. The Punchboard after results are posted. The committed data for the receipts are revealed, and voters can check the Punchboard to ensure their vote made it to the final tally.

populate the Intermediate Position and Real Vote columns, improperly publishing a result in either column would result in an overwhelming probability of being caught. Illustrations of both column choices are given in Figure 3.7 and Figure 3.8.

Alternate Audit Instead of the entire column, the columns on each row could be individually chosen. This may be undesireable as it reduces the privacy set by half. That is, there become two distinct groups of ballots: those who's left column was opened, and those whose right column was opened, and we can group the results into these two sets.

Recount or Reaudit If some impropriety is found in the auditing process, the Post-election audit can be rerun by producing new D tables, publishing the results, and having the auditors pick from the new tables. It is not possible to rerun the first two audits, because

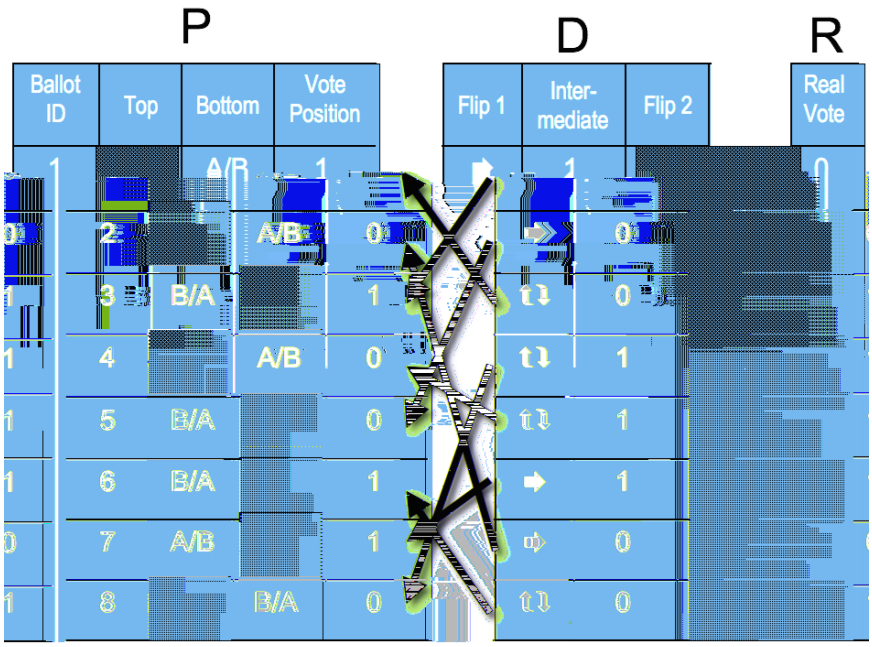

FIG. 3.7. **Post-Election Audited Punchboard**. A final version of the Punchboard after the post election audit. In this version, the auditor chooses the left side of the D table and the Trustees reveals it. Now, we can see how the Vote Position column corresponds to the Intermediate Position column and verify that every Vote was accurately recorded in the Intermediate Position Column.

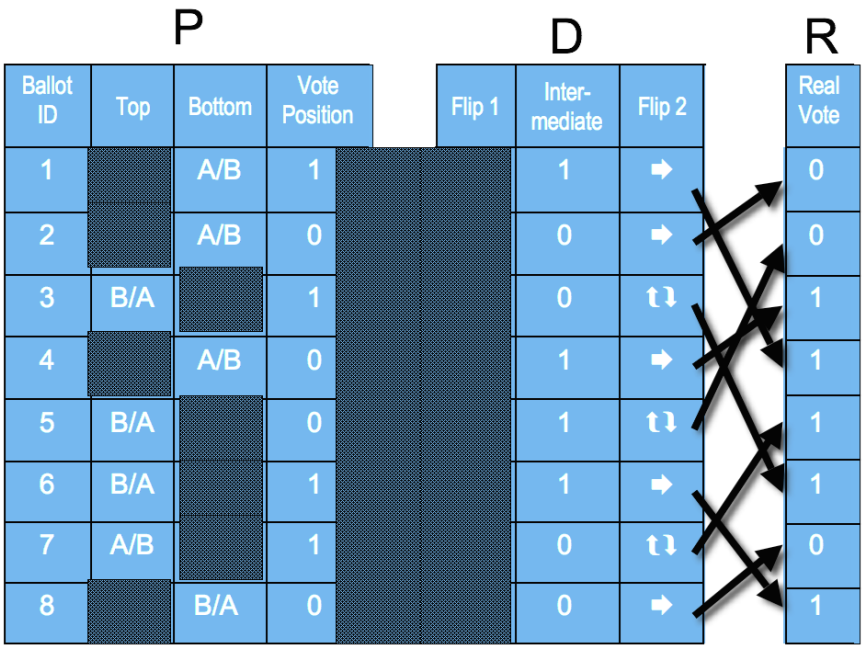

FIG. 3.8. **Alternative Post-Election Audited Punchboard**. A final version of the Punchboard after the post election audit. In this version, the auditor chooses the right side of the D table and the Trustees reveals it. Now, we can see how the Intermediate Position column corresponds to the Real Vote column and verify that every Vote was accurately recorded from the Intermediate to the Real Vote column.

they are integrity checks on the printed ballots and voted positions. Because of the counting protocol, any counting mistakes can be detected in the same way as before.

3.6 Security Notes

Kevin Fisher conducted the first security analysis of Punchscan [24]. His analysis indicated three security concerns:

1. **Mark Coercion**. While the ballots in Punchscan are coercion resistant, an attacker can take advantage of the positional mark information to carry out some attacks on the system.

2. **Trust in Third-Party Printers.** A third-party printer could potentially reveal information necessary to decrypt ballots outside of the decryption process.

3. **Ballot Assurance.** Punchscan does not provide protection against forgery of ballots. A voter can detect if her vote is not counted or is misprinted, but she does not necessarily know if the ballot she recieves is a valid ballot.

Chapter 4

A TRUSTED WORKSTATION BETWEEN MUTUALLY DISTRUSTING PARTIES

Election trustees use a special workstation in the Punchscan system to calculate election results and respond to audit inquiries. It is trusted in the sense that it protects the confidentiality of the data, that is, the link between a posted ballot receipt and its corresponding decryption in the results table. The confidentiality here is closely related to voter privacy, and it is important in the Punchscan system. Since the workstation is the only computer responsible for this privacy, a high threshold is set on its security.

The meaning of *trusted* in this setting is different from the traditional meaning where it might also mean to be trusted with integrity of the data. The integrity of the data the workstation sees is guaranteed by the auditing processes in the system.

Also, this is not a system where all adversaries are supposed to be prevented from using the system. On the contrary, the trustees who use the system are expected to have adversarial relationships (i.e. be members of different political parties). The trusted workstation they use can be thought of as the referee they all agree to trust to make correct decisions and enforce certain rules. From this unusual situation stems a unique set of requirements.

In this chapter we outline the requirements and propose an architecture for Punch-

scan's trusted workstation. To meet our requirements, we introduce a threshhold *secret sharing scheme (SSS)* we call *user contributed secret sharing (UCSS)* and a software validation routine.

The software validation routine requires that each election trustee bring his own read-only copy of the software. Each copy is chosen at random to carry out a validation process that verifies that all copies are identical. Also, UCSS lets each user contribute entropy to a master key which is stored in a way that allows a subset of members to decrypt it when all members are not present.

In the following section we discuss the requirements of the trusted workstation and give our proposed architecture. Afterwards, we consider alternative approaches to the chosen trusted workstation approach. In the final section we analyze our design and discuss alternative choices we could make when building the system.

4.1 Requirements

The trusted workstation is used by a group of users collectively known as the Election Authority (EA). Each member of this group, an election trustee, should be able to have confidence that the software running on the workstation is the correct software that was published online before the election. If each trustee can be sure the software used is the correct software, he can enforce other actions (such as not storing or transmitting the secret key) that maintain security in the system.

The trustees should be able to use the system only if enough show up after the system is initially configured, and the number required to be present should be set by the EA at the beginning of the election. The workstation should be able to generate and protect cryptographic keys in the system. No number of trustees under the threshhold should be able to retrieve the keys.

The trusted workstation should not permanently store or transmit secret information. Its hardware should only contain video, memory, BIOS, CPU, and support for basic input devices (i.e. keyboard, mouse). Its ability to store and transmit data should be limited to interfacing with a physical external source that must be plugged into the workstation.

The software itself should be deterministic. This requirement is twofold — we can check a suspicious workstation by performing the same computation on a different workstation and catastrophic data loss should not be a problem if the EA can meet and reperform calculations with the software on a different workstation.

All source code for the Workstation's operating system and user applications should be open and published along with any derivatives, including compiled binaries and disk images. All published code and binary data should be accompanied by a public hash value and the steps necessary to reconstruct any derivative from the original source code. This requirement allows anyone, including trustees or their representatives, to use publicly available tools to examine, build, test and verify the software to be run on the Trusted Workstation.

To our knowledge, our requirement for a group of untrusted users to validate that the expected software is running on a machine is unique. It follows that if the system can meet this requirement then it should be able to enforce any software-based restrictions necessary.

The threshhold requirement is also unique because, while normal systems must authenticate users before operation, the workstation must authenticate a group of users, generate a cryptographic key, and also prevent any group of users smaller than the threshold from retrieving that key. The system lacks persistent storage, and no assumption is made that one trustee will not be able to read data on external storage when the workstation is not in operation. The situation is similar to secret sharing as explained in 2.2.6 except that there is no dealer to split up a preexisting key, and all the would-be recipients of the shares are at the same place at the same time. The workstation also prevents each trustee from

in that sense it will always be faster. If a teller is created that is faster than the Punchboard, we could swap it in, and 1 teller would be faster than several. Again, there is no reason to use more than one teller, because the last teller provides the privacy in a system with printed ballots.

2. **Communications Requirements.** The tellers need to communicate with each other, and, depending on implementation, possibly with the polling places. In practice this communication would require an Internet connection. The Punchscan system is off-line in the sense that data are hand-carried to a special room, operations are performed on a closed system, and then hand-carried back out. While the same thing can be done with a mixnet architecture, it would introduce significant overhead.

3. **Reliability and Recoverability.** If one teller refuses to perform their operations faithfully, the system ceases to work.

4. **Trusted Hardware/Software.** Having multiple tellers can help with the trusted hardware and software problem, but it is not as effective as it appears. One security assumption in the teller model is the reliance on different teller implementations. However, due to software complexity and certification requirements, it is likely that the implementations would be homogeneous. If an attacker can simply do the same thing to all tellers, it does not greatly increase cost as might be expected.

Note that a mixnet that does not use printed ballots must use a computer located at the polling site that will encrypt the choices of each voter by using a public key and will print out a receipt of the encryption that is posted online. This computer would know the choices of each voter that uses it, could print information on the receipt that indicated those choices, may not encrypt the receipt properly, and may communicate with a third party in another way. Using a printed ballot reduces the number of machines trusted with privacy

to the trusted workstation and the printers. The printed ballots enable an audit to detect that they were printed correctly, but must be procedurally protected.

These differences coincide with the prevailing view that it is the responsibility of the EA to protect the identity of each voter. Note that the EA already is responsible for protecting privacy in existing systems: For paper based systems, the election authority is charged with protecting the paper ballots, all of which contain identifying marks (finger prints, write-ins, etc). For both paper based and digital systems, we expect the EA to keep cameras and other recording devices out of the polling area.

4.2 Assumptions

We assume that all hardware used is benign. This assumption can be enforced by nondeterministically choosing which computer to use: assuming a list of computer stores within 50 miles of the EA. The EA will randomly choose what store and machine from that store to use, bring it back to the EA, and (optionally) destroy that machine after it is used. This assumption is more difficult in an implementation that does not use commercial of the shelf (COTS) computers. In that situation, we assume that members of the EA or their delegates have ample time to inspect the device for a malicious of hardware or a malicious bios and are able to correct such problems before use.

We assume that a majority, or the threshhold set by the EA, of trustees are honest. This assumption follows from our threshhold requirement. If a majority of the trustees are dishonest, then they can release information to decode receipts. We also assume that trustees do not trust each other.

We minimize our assumptions on the EA's ability to protect the system and the data coming into and out of the system. We assume that the EA is able to transfer ballot information confidentially to the printer and delete it appropriately, but we do not assume

that any storage necessary between meetings of the EA can also be kept confidential. The difference here is subtle. In the first assumption, two election officials of opposing interests are able to prevent each other from leaking information. By contrast, in the second assumption, storage needed between meetings might be stored in a lock or safe, and may not be always be guarded such that all individuals are prevented from accessing the data until the next meeting. Thus, the more general assumption is that the EA will be able to protect data for short periods of time, but not longer than for one day.

These assumptions are important in the realization of our architecture. In the next section, we discuss the architecture of our system.

4.3 Architecture

There are two major components of our trusted workstation: a bootable operating system on write-protected storage, and software to be run on the system that enforces the threshhold requirement. Running the system on write-protected media is necessary for the system to support validation. Each trustee needs to know that the correct software is running, so our proposed solution is that each trustee separately use the data made publicly available to load it onto bootable media independently and bring it with them to be checked at the trustee meeting. Once there, they can compare the media to make sure they have all brought the correct software, and randomly choose which copy to use.

The next section explains the properties of the bootable operating system. After that, we discuss the responsibilities of the software and its five major tasks: Software Validation, Key Generation, *User Contributed Secret Sharing (UCSS)*, Management of Public/Private Data, and Creation of a Signed Audit Log.

4.3.1 Bootable Operating System

The bootable operating system must be able to run the software and must be able to detect video hardware on any computer on which it is run. It cannot save any settings, because it must run off of a write-protected storage device. A modified LiveCD distribution as described in A could be used for this purpose.

Unfortunately, such a solution creates problems for software validation. Trustees need to check each copy of the software with their own copy. This process requires a CD drive for each trustee, or that the software check all the other trustees software one CD at a time. Additionally, different CD writing software might create small changes on the disks. Thus, we propose usage of a USB drive with a write-protect switch. We prefer USB drives in favor of recordable CDs for several other reasons:

1. USB drives are reusable, so they have less impact on development and testing.
2. USB drives give us flexibility to allow other drives to be used for output produced by the core engine and an audit log.
3. USB drives are generally faster than CD ROM drives, and there is no worry of the drive spinning down during use.
4. Most computers come with multiple USB ports; this is not the case with CD ROM drives.

4.3.2 Software Tasks

The software runs automatically after the operating system boots and user interface is loaded. After loading, it will perform operations in the following order:

1. Software Validation

(a) Request all removable media to be tested be inserted

(b) Hash and compare all removable media, report results

2. Request Public/Private Data Storage Locations

3. User Contributed Secret Sharing

   (a) Gather Usernames/Passphrases

   (b) Generate or Retrieve Necessary Public/Private and/or Secret Keys

   (c) Save Public/Private and Secret Key Data, if necessary

4. Run Trusted Process

   (a) Ensure required data exists for selected operation

   (b) Run the appropriate piece of the Core Engine.

   (c) Save data to preselected Public/Private Data Storage Locations

5. Sign Audit Log with Retrieved Private Key and save it to the Public storage location.

If Step 3 is not successful, it is impossible to perform election calculations, and impossible to produce a signed audit log. Software verification is performed multiple times, but will only be reported once on the signed audit log because verification requires that the computer be booted from different copies of the software. Last, if there are problems in any of the sub-steps, the software should allow the users to repeat that step from the beginning.

While most of these steps are self-explanatory, there are two steps that stand out: 1 and 3. Software validation is the process we designed for trustees to verify that they have all brought the same software with them to the meeting. *User Contributed Secret Sharing (UCSS)* is a secret sharing system we developed to all trustees to bring their own entropy with them to create and share a master key with a threshholding ability for later access of that key. We describe each of these core contributions in the next two sections.

4.4 Software Validation

In order to perform the software validation participants will perform the following actions:

1. Switch all of their disks to read-only.
2. Pick a participant's media that has not yet been booted from and boot from it.
3. Plug in and hash all the other participant's disks, verifying that all the hashes are correct.
4. Return to Step 2 until there are no unbooted participant disks.

Unless all of the software is corrupt or the validation is compromised, this process ensures that any malicious software is discovered before any protected operations are performed or any sensitive data is entered into the system. If the software is found to be corrupt, that trustee should have the option of getting a new copy or choosing to trust the copy of another trustee.

4.5 User Contributed Secret Sharing

The UCSS step will produce or retrieve the public/private key pair and a secret key that the engine software needs. The "share" that each trustee brings with him is a username/passphrase combination. All participants must be available the first time they meet. Each subsequent time, if all the participants are present, the software will rerun the key generation step. If the necessary threshhold is present, it will perform key retrieval (secret sharing). The software determines which to use based on how many participants enter their shares.

$E_{H(user1,pass1)}(c)$
$E_{H(user2,pass2)}(c)$
$E_{H(user3,pass3)}(c)$
$E_{H(user4,pass4)}(c)$
$E_{H(user5,pass5)}(c)$
...

Table 4.1. **Format of UCSS passwd file**. E is the encryption function, H is the hash function, and c is the public constant.

Entering Shares. Each of the participants enters a username and passphrase. If this is not the first time they have met, each username and passphrase is checked against a passphrase database. The passphrase database is an encryption of a public constant with a key generated by a hash of their usernames and passphrases concatenated together. If this check fails, an error is recorded in the audit log and that username and passphrase pair are not included in any secret sharing or key generation. If this is the first time the group has met, the public constant is encrypted using the hash of each username and passphrase concatenated together. Before moving on to key generation, the software will save this file in the Private storage directory chosen by the participants.

Key Generation and Retrieval. If all participants were present and their usernames and passphrases are correct, or if this is the first time the group has met, then we perform key generation as follows: All usernames are concatenated together, and all passphrases are concatenated together, then the concatenation of usernames and passphrases are concatenated together. This value is then hashed, and the output of the hash is the secret key which will be used by the engine. This value is also used to seed a Pseudo Random Number Generator (PRNG) which is used to generate a Public/Private key pair. As an example, for

five users $u_1, ..., u_5$ with passphrases $p_1, ..., p_5$ and the hash function H, the secret key S is:

$$S = H(u_1u_2u_3u_4u_5, p_1p_2p_3p_4p_5) \tag{4.1}$$

To store the keys, the software needs a threshold k set by the users. This value represents the difference between the number of participants n and the number needed to regenerate a key S. We then encrypt our private key with our secret key. Finally, for each subset of participants of size $n - k$, we hash the usernames and passphrases, and use that to encrypt the master key. All of these encrypted keys are stored in a document describing what participants' shares were used to encrypt that version of the key. To retrieve the key, we determine what participants entered keys by their order in the passphrase file, decrypt the corresponding encrypted key from the file storing all possible encryptions, and use our decrypted secret key to decrypt the private key. For example, if we have three users $u_1, ..., u_3$ with passphrases $p_1, ..., p_3$, a threshhold value of $k = 1$, an encryption function E and a hash function H, we will have the following encryptions of the secret key S:

$$E_{H(u_1u_2,p_1p_2)}(S) \tag{4.2}$$

$$E_{H(u_1u_3,p_1p_3)}(S) \tag{4.3}$$

$$E_{H(u_2u_3,p_2p_3)}(S) \tag{4.4}$$

4.5.1 Using Other Secret Sharing Schemes

Because the software validation protocol gives assurance that the correct software is running, we could potentially use the validation scheme with any secret sharing scheme such as those discussed in 2.2.6. To maintain the user contributed nature of the system, we could use entropy provided by the trustees as the secret to be shared by any secret sharing scheme, or as a seed to the random number generator that creates the shared secret.

If we used another secret sharing scheme, the only change from the trustee perspective would be that each trustee would bring recordable media to the first meeting in order to record his share of the secret. He is then responsible for keeping that media safe and bringing it to subsequent meetings.

One way to help keep shares from being stolen from trustees might be to use each trustee's passphrase information to encrypt the share on his media. However, if left unencrypted, simply producing enough of the shares would be enough to perform operations, and no passphrase would be necessary during subsequent meetings.

The advantage of an alternate scheme are space efficiency. As we discuss in 4.6, our scheme is simple to implement, but it requires much more storage than other secret sharing schemes. Our scheme also has a significant advantage because it needs no physical token, which could be lost or stolen in a traditional secret sharing scheme.

4.6 Analysis

The software validation protocol in tandem with the secret sharing scheme offer significant protection of the underlying Punchscan voting system. Defeating them requires a high level of technical sophistication.

4.6.1 Defeating the Secret Sharing

The security of the secret sharing scheme relies on the ability of each trustee to keep his passphrase a secret and the properties of the cryptographic functions used in the system. If enough of the trustees cannot keep their passphrase secret the scheme can be defeated. Likewise, if one of the cryptographic functions has a defect, it may also allow an attacker who gains access to the storage device to defeat the system.

The most significant weakness with regard to the cryptographic functions is that the

constant value must be known. This knowledge allows an attacker to perform known plaintext known ciphertext attack on the encryption function. If an attacker succeeds at this attack, he will know $H(username, passphrase)$ and must additionally break the hash function to reveal the passphrase. To completely defeat the system, he must perform this attack on the minimum threshhold number of stored pass phrases required by the system.

Alternatively, an attacker may brute force attack the system. Using a secure hash and encryption function like this scheme only introduces a linear amount of extra work over using a cryptographic hash only. The scheme does, however, protect against weaknesses that may be present in the hash function because the output of the hash function is not shown to the attacker.

Using an alternate secret sharing scheme as explained in 4.5.1 would avoid these problems but would create others. It would rely on the ability of a trustee to protect a physical device, and the data represented on that device could be copied without knowledge of the trustee. Encrypting with the original trustee password would be sufficient protection from this threat.

4.6.2 Defeating the Software Validation

The software validation could be defeated by a slight of hand magician or special hardware. We minimize the impact by permitting situations in which the hardware used can be chosen at random, and each trustee handles his own media brought to the meeting. Additional procedural safeguards could be used by the EA.

4.7 Discussion

Unfortunately, the software validation strategy can be rather tedious and slow, especially if there are a lot of participants, but there are ways that it could be sped up:

1. A "fast check" mode, in which a statistically significant portion of the drive (instead of the entire drive) is randomly tested to ensure a certain probability of finding a bad byte.

2. Faster USB drives or improved bus speed.

3. Hardware support for machines with multiple processors or other hardware that provides performance enhancements.

4. A compact linux distribution. We could reduce the size of our bootable USB drives to between 50 and 100 megabytes, making the validation process very fast.

The secret sharing scheme is far from space efficient requiring $\binom{n}{n-k}$ copies of the key, but it does allow us to achieve the following properties:

1. It is a threshold secret sharing scheme.

2. We can detect which participant was cheating, not just that cheating has taken place.

3. Each participant brings his own share and contributes to the key generation process.

4. The keys are regenerateable if the drive containing the encrypted passphrase file and other data are lost or one less than the threshold value of participants is unable or unwilling to produce their share.

5. Only one storage device needs to be utilized by the participants to depend on key retrieval, if necessary. Most schemes would require k of the users to bring their storage devices, we only require that they remember a username and passphrase.

Chapter 5

INDEPENDENT BALLOT SHEETS

In Punchscan [5, 25, 45], any entity with access to both ballot sheets can violate voter privacy by recording the serial number and the permutations on each sheet. Inspired by the concept of a binary weapon,[1] we explore the idea to create a Punchscan ballot in the polling place by separately combining independently printed ballot halves, each with a separate serial number, with the hope of reducing required trust in the printers and thereby enhancing ballot privacy.

We present a strategy that prints top and bottom ballot sheets at printers in different geographic locations and keeps them separate until each voter selects, or is given, an arbitrary top and bottom sheet. Then, the voter combines the sheets to vote. An *Independent Ballot Sheet (IBS)* ballot is illustrated in Figure 5.1.

Traditional Punchscan ballot sheets can be printed separately, but polling places must match the serial numbers on the top and bottom sheets when they give the sheets to each voter. Instead of printers, the workers who combine the sheets become the entity capable of violating voter privacy, because they must match up the serial numbers on the top and bottom sheets. This matching process also adds other forms of overhead: the separate printers should print and package the sheets in the same order, election officials must coordinate

[1] In a binary chemical weapon, two chemicals are separately stored, each safe by itself. Only when combined do these two ingredients form a dangerous substance.

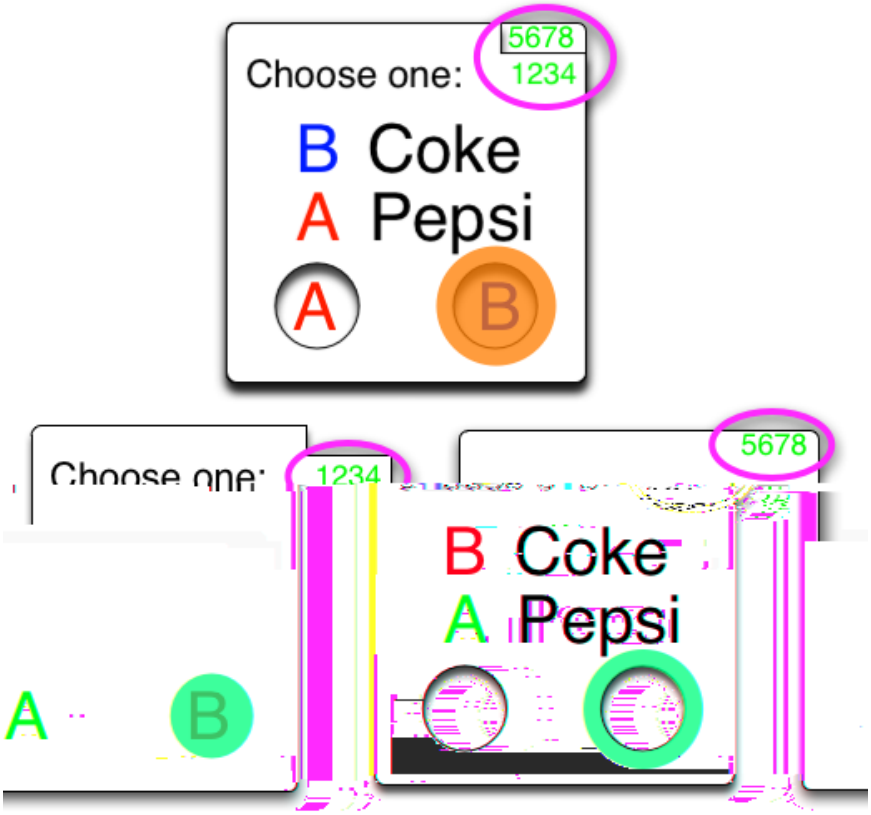

FIG. 5.1. **IBS Ballot.** Ballot sheets in the IBS punchboard do not have the same serial numbers and can be combined arbitrarily.

with the printers to make sure the top set and bottom set of sheets are sent to the same polling place, and extra manpower is required to combine the sheets.

In both systems, after marking a two-sheet ballot, the voter destroys one of the sheets. In Punchscan IBS, the serial number of the destroyed sheet is copied onto the surviving sheet. For ballot privacy, in both systems it is important that an adversary cannot determine the random permutations on the destroyed sheet.

If each ballot sheet is packaged such that serial numbers are shown without revealing the contents of the sheet, then the privacy properties of Punchscan IBS with separate printers and traditional Punchscan with separate printers are similar, but Punchscan IBS offers greater simplicity and flexibility in printing and distributing ballots. Punchscan IBS also has a greater resistance to disruption than what can be accomplished in the original system. Mistakes in shipping do not necessarily cause problems in IBS, so long as there are some top and bottom sheets available. Additionally, in the event of a ballot shortage, poll workers can create more valid Punchscan ballots using a copier machine.

5.1 PageScan

Fisher [24] was the first to explore changing the Punchscan system to support combining top and bottom sheets with independently chosen serial numbers with his proposal of PageScan. We found PageScan to have a ballot privacy flaw and some inefficiencies when compared with IBS. IBS can be thought of as an improved continuation of Fisher's work, and this section explains PageScan to give the reader better understanding of IBS.

5.1.1 The PageScan Protocol

PageScan uses a similar structure to the Punchboard which Fisher calls the PageBoard. The complete, unredacted form of the PageBoard in shown in Figure 5.2. The format of

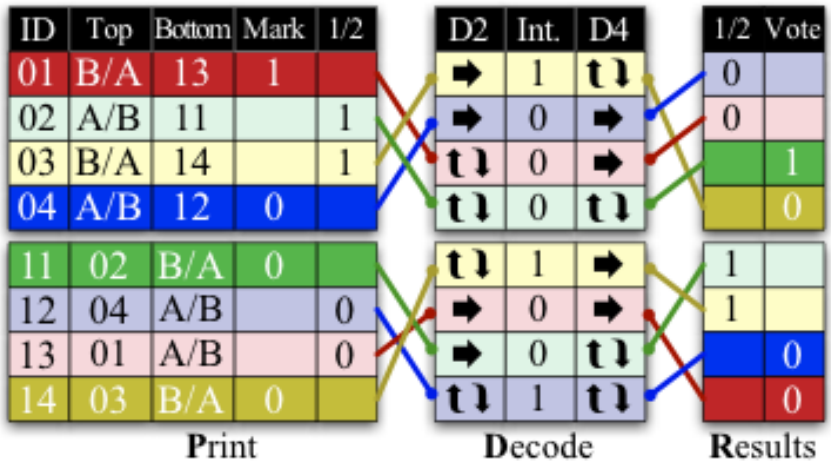

FIG. 5.2. **Complete PageBoard.** An unredacted form of the PageBoard which illustrates the computations performed by the Trustee Workstation to decode a ballot in which two sheets of different serial numbers were arbitrarily combined. The column of the discarded sheet lists the mark position, it is then process through the table and the intermediary result is published next to the receipt sheet.

the PageBoard is similar to that of the Punchboard, with the exception that the top sheets and bottom sheets of the ballot each share their own table, and the ID number each sheet is paired with is entered into the bottom column for the top sheet table, and the top column for the bottom sheet table. The operations and auditing processes of the PageBoard are also very similar to those of the Punchboard.

Processing of a ballot through the PageBoard works as follows:

1. The ID number of the discarded sheet is copied to the receipt sheet and the relationship between the two sheets is placed into the system.

2. The marked position is entered into the Mark column in the row of the discarded sheet for each ballot.

3. The values entered into the Mark column are then processed through the Decode table and published in the 1/2 column in the Results table.

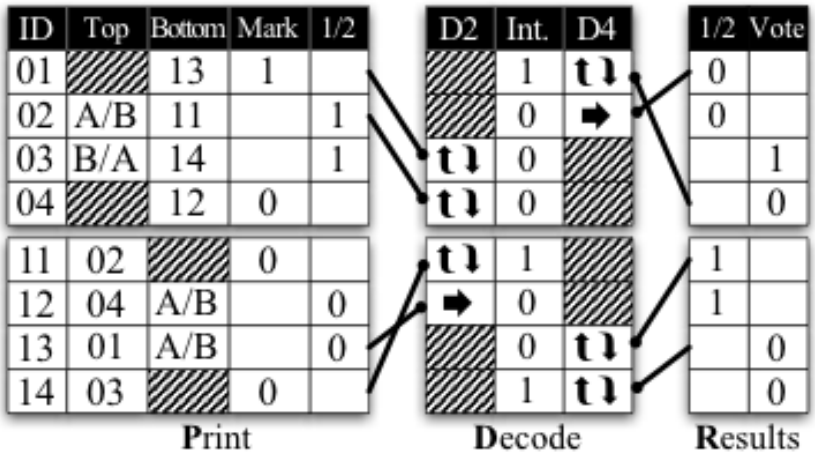

FIG. 5.3. **Audited PageBoard**. The audited punchboard is what is shown to the public after auditors make their selections to reveal parts of the ballot decoding process.

4. The values in the 1/2 column in the Results table are copied to the 1/2 column in the Print table in the row corresponding to the receipt sheet.

5. The values in the 1/2 column of the Print table are processed through the Decode table a second time and Results are calculated.

Auditing in the PageBoard works the same way as it did in the Punchboard as described in Chapter 3. After results are posted, auditors make their selections between the left or right halves of the Decode table for each row, and the operation performed is revealed, then anyone can check to make sure the operation performed matches the result posted in either the intermediate or results columns. The PageBoard is only different in that ballots are processed twice through the table, and the 1/2 column serves as a result column for auditing the first time through the table.

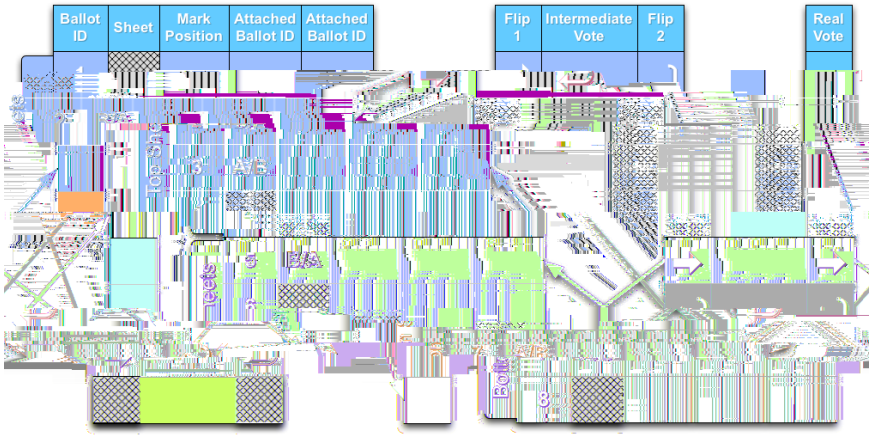

FIG. 5.4. **Pre-Election**. The Punchboard after the pre-election audit. The data in half of the rows are posted so the public can verify that the Punchboard is well-formed.

5.1.2 Breaking Ballot Privacy in the PageScan PageBoard

The problem with ballot privacy in PageScan is that the receipt reveals all information necessary to decode it via the table. Thus, by publishing the 1/2 result next to the receipts, we can combine these two pieces of information and the ID of the discarded sheet in order to determine the vote on that receipt.

In order to fix this problem, The decryption should be done in the opposite way by posting the marks next to the receipt row, decrypting, and posting the result next to the discarded sheet. Observe, however, that decryption of the receipt is unnecessary. Alternatively, we can combine receipt information and the mark position together and include it while decoding the discarded sheet. This action is similar to what is done in the IBS punchboard.

5.2 The Independent Ballot Sheet Punchboard

We now present a modification of Punchscan to support *Independent Ballot Sheets (IBS)* and show that our method allows us to maintain auditing and integrity properties that

are at least as strong as those in traditional Punchscan. Our modification does not change the way people vote, but it does require the sheets to be combined in the polling booth and that the serial number of the destroyed sheet be recorded onto the receipt. It also changes the way the Punchboard is structured and used, and the meaning of its tables.

In the original system, both the Print and Decrypt tables had combined sheets represented in each row, but now each row represents a single half-sheet. The structure of the Punchboard Decrypt (D) and Results (R) tables remain the same, but the Print (P) table changes and the number of rows in all of the tables are doubled. The new P table has 4 columns. The first column, $P1$, records letter order on either a top or bottom sheet. $P2$ records the position marked by the voter after if that sheet is taken by the voter as a receipt. $P3$ records the sheet that the current sheet was paired with when it was used. $P4$ records the mark position after the value in the receipt, $P1$, is removed from the recorded mark position, $P2$.

To generate the Punchboard, let n be the number of ballots that will be available for voters. The election authority (EA) then generates $2n$ virtual top pages and $2n$ virtual bottom pages and puts them in the P table. For simplicity, we will assume the top pages are in the first rows of the P table (positions 1 to $2n$, therefore having serial numbers from 1 to $2n$) and the bottom pages are in last rows (positions $2n + 1$ to $4n$, therefore having serial numbers from $2n + 1$ to $4n$). The EA creates a D table where each row will correspond to a row in P. Therefore half of the rows in D will correspond to top pages and the other half to bottom pages. The EA commits to the rows that this creates, just as in the previous punchboard. The rows in D are then shuffled and the commitments to the rows are published.

Pre-Election Audit. Figure 5.4 illustrates the pre-election audit. In the pre-election audit, the auditors choose n top sheets and n bottom sheets from the P table. The election authority opens the rows in P and the corresponding rows in D. Anyone can check the

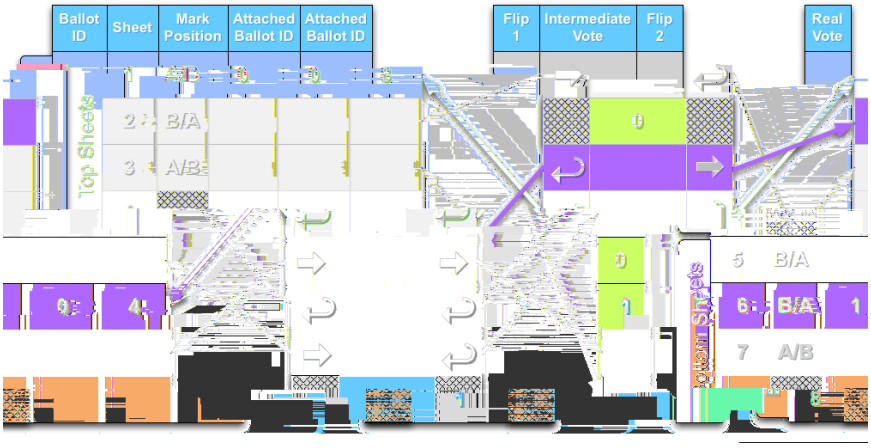

FIG. 5.5. **Results**. The Punchboard after results are posted. Half of the Mark, Intermediate, and Results columns are populated to give unaudited results of the election. Note that sheet 003 was paired with sheet 005, and the number 5 appears in the paired top column. Likewise, sheet 008 was paired with top sheet 001, and it appears in 8's paired sheet column.

commitments and the fact that $P1 = D2 \oplus D4$ ($\oplus$ meaning the commutative composition operation), *i.e.* that the value to be printed matches the sum of the two inversions. This slightly altered process produces the same result as before, with half of the possible number of ballots being discarded to verify that the Punchboard is well-formed.

Posting Results. At this point, the ballot pages are printed and any top page can be combined with any bottom page. The voting procedure is the same as before. In addition to the voter marks and the serial number, the receipt must also contain the serial number of the sheet it was paired with that has been destroyed. We could do this by not destroying the serial number of the destroyed sheet, or by copying the serial number to the receipt and signing it for authenticity.

Now we have a correspondence between the receipt $P1$ and the sheet it was paired with, $P3$. The receipt the voter took home is revealed by $P1$, and the discarded sheet serial number is $P3$. $P4$ represents the ballot only taking the destroyed sheet into account. That is, $P4 = P2 \oplus P1$ for each row in P with a populated $P2$. For example, if the mark is left

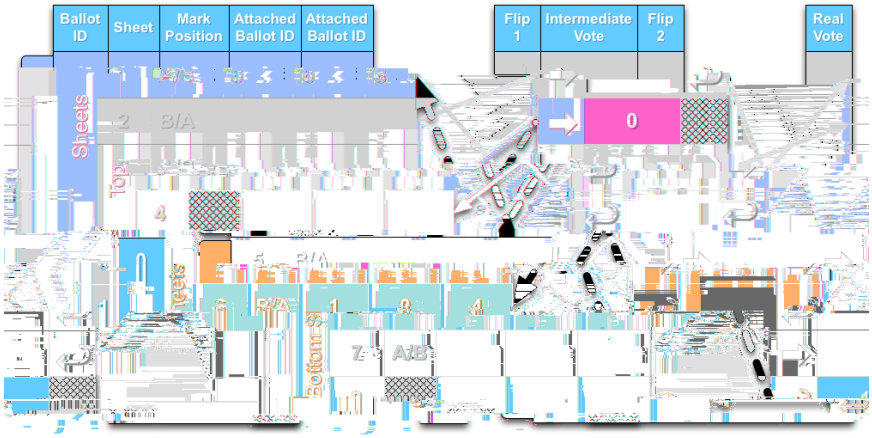

FIG. 5.6. **Post-Election Audit**. The Punchboard after the post-election audit. Data to the left or right of the Intermediate Position of the Decrypt table are revealed to audit the results of the election.

and the receipt is an inverting page, the $P4$ column contains a right mark. Once the receipts are published, anyone can compute $P4$, as illustrated in Figure 5.5. Also during this time, the EA computes $D3$ and R, filling in the Intermediary and Results values. Note that when counting, $D1$ now points to the value in $P3$, not $P1$ as in the pre-election audit.

As before, each voter is able to verify her ballot is included in the tally and is well-formed, and everyone is able to verify that $P4$ was computed correctly and that the revealed data match commitments.

Post-Election Audit. Figure 5.6 illustrates the post-election audit. The figure shows selected row halves of either the two columns left or right of $D3$ being posted. In the actual method, where multiple D tables are published, the entire columns to the left or right are posted. The post-election audit remains virtually unchanged from the original scheme. The major difference is that the audits reveal information that is not necessarily needed to verify integrity of the system because that information is revealed in the results phase when receipt values are posted. This information is denoted by the blank mark in the intermediate and results cells in the tables.

5.3 Analysis

The primary innovation of IBS is that it enables complete ballot information to be stored in separate places until the voter combines the two sheets to vote. It accomplishes this without requiring tracking of a corresponding sheet by election officials, and offers several advantages and disadvantages.

5.3.1 Privacy

In the original system pages are printed and stored in the same location. If an attacker gains access to the ballots he can violate voter privacy as he can access both sheets of the ballot. The separation of the ballot sheets until the voter combines them in IBS means an attacker must intercept both sets of sheets to violate privacy. If he intercepts only one set, he has a 50% chance of violating privacy, and this offers a privacy advantage over the original system.

With special packaging and tracking of the sheets, the original system could potentially use independent printers as IBS does. In this case, our modification has a similar effect on voter privacy as the original system. To see the differences, consider the following three cases:

1. Punchscan with 1 printer.
2. Punchscan with 2 printers.
3. IBS with 2 printers.

In Case 1, ballot sheets are created, printed, and stored together. If any of these sheets are compromised, an attacker knows the information on both sheets and can determine the meaning of marks on any receipt. Cases 2 and 3 offer a distinct advantage. By separately

printing the top and bottom sheets, if the voter takes home an uncompromised sheet, the attack does not succeed. On the other hand, a coercion attack may still work if the victim is unwilling to risk that the coercer has compromised the correct sheet. Case 3 provides some extra flexibility over case 2 yielding a marginal advantage, because there is no need to ensure that matching ballot serial numbers are combined.

5.3.2 Reliability and Logistical Properties

IBS has several rreliabilityand logistical advantages over the original system when using independent printers. It does not require a packaging where the serial number is visible but the rest of the sheet is not. Thus, packaging can be done in bulk, and voters, instead of being given two pre-matched sheets, can pull a sheet out of two separate piles of ballots.

Election officials may easily arrange for different printers to print various top or bottom sheets. The more printers used, the less likely an attacker will be able to conduct a targeted attack because he will not be able to ensure that the target voter will receive and choose to destroy a ballot sheet that they have compromised. Additionally, IBS does not require election officials to predetermine what serial numbers are sent to specific polling places, only that they specify how many sheets and of what type to send.

Shipments of ballots do not need to be labeled with what serial numbers are contained in the shipment. Likewise, mistakes in shipping do not necessarily cause a disruption. A polling place sent incorrect serial numbers or missing some sheets of a certain type can continue to operate as long as they have sheets of both types available.

Creating New Ballots On Demand One possibility of IBS is that it can enable poll workers to create new ballots with a copier machine, but use of this ability can drastically affect the privacy of the system. In the original system, if the number of voters exceeds the

number of available ballots poll workers must contact nearby polling places or the election trustees to get more ballots. This situation can potentially turn into a denial of service problem because extra ballots may not be available, it may take too much time to deliver the extra ballots, or new ballots may need to be created in an additional meeting of the election trustees.

To create new ballots, poll workers will choose groupings of i *public* and j *private* sheets. The *public* sheets will become receipts, and the *private* sheets will be destroyed. For each of the j *private* sheets, election officials make a copy of each *public* sheet and pair it with a copy of the *private* sheet. This process creates $i * j$ new ballots, or, in other words, i ballots for each *private* sheet. As the copied pages will not have the required drilled holes, voters must look at the *private* sheet, and appropriately mark the *public* sheet.

To count these copied ballots, extra rows must be added to the P, D, and R table for each unique top and bottom sheet pair. Essentially, they must be copies of the row for the *private* sheet that was destroyed. The D table can, optionally, be recomputed which maintains the privacy set of the system. If rows are simply added without the entire table being recomputed the hashes must be uniquely keyed for each row. This affects the privacy set by reducing it to the number of *public* sheets that were attached to the paired *private* sheet in the table.

Unfortunately, this process has significant drawbacks. Most significant is that the poll workers must violate the privacy of the system and now become a trusted part of it. Revealing data on one *private* sheet would violate the privacy of several voters. Poll workers would also control what sheet is chosen, and a voter would no longer have the opportunity to observe the selection process as before. The process is also prone to error, although we think that with proper directions and training, it could be feasibly performed by team of poll workers.

In a traditional optical scan system, one poll worker can take a ballot and make an

unlimited number of copies. Creating new ballots in IBS, while possible, is more complicated than this process and can only create $i * j$ new ballots. However, it is still feasible. Creating new ballots from a set of 100 top and bottom sheets can create up to $10,000$ new ballots. By contrast the original system can only provide a traditional optical scan ballot to handle this problem.

5.3.3 Printing

Punchscan IBS prints on one sheet of paper at a time, creating some challenges and benefits. In the original system, tolerance for aligning the sheets must be very low to avoid printing bottom sheet information on the top sheet. Printing the sheets separately eliminates this possibility. The tolerance is higher in IBS, but using different printers increases the chance of printing errors that produce unusable ballots. On the bottom sheet, too much skew can misalign letters from their corresponding top sheet holes.

The need to package each sheet separately increases cost. At the same time, cost is marginally reduced by not having to fold the sheets as is done in the original system. Also, feeding one sheet is more reliable than feeding two sheets into the printer.

Again, the traditional system can achieve some of these benefits if it, too, uses independent printers, but in IBS it does not matter what order the ballots are printed. This gives IBS a distinct advantage. In the traditional system, the ballots would have to be printed and packaged in order. IBS is not disrupted if a printer skips ballots, but the traditional system would cause some confusion at the polling place when workers tried to match up serial numbers. This feature also enables printers to purposefully print and ship their sheets at random.

5.3.4 Usability

The ballot sheets must be combined, and the serial number of the destroyed sheet must be copied over to the receipt. In the original system, the ballot sheets are already joined and each sheet has the same serial number. If the original system uses independent printers, then the paired sheets must be combined according to their serial number. IBS offers a slight advantage in this case, because poll workers and voters do not need to make sure that the serial numbers match.

Combining the ballots and ttransferringthe serial number to the receipt sheet do cause additional complexity. We believe that there is no reason it cannot be done in a mechanically robust way. For example, the serial numbers could be perforated such that they could easily be removed and stapled or glued to the receipt sheet. Combination of the sheets could be aided with a special clipboard similar to the one currently being used in the Punchscan system that holds the sheets in place for voting [5].

5.3.5 Other Properties

IBS has two minor properties that are different from the original system. First, it doubles the number of rows in every table of the original system, and adds two columns. However, the auditing and privacy properties of the original scheme are maintained.

Second, the change to a single permutation per row of the table closely resembles the single permutation operations of SureVote [15] and Prêt à Voter [17]. These similarities show how the Punchboard could be used interchangeably with these systems.

5.4 IBS in Context

IBS enables election officials easily to print the top and bottom sheets separately, complicating attacks on ballot privacy. By contrast, such a printing strategy is not possible

with Prêt à Voter because it has only one sheet.

Other variations to Punchscan might also be worth investigating, including printing ballots at each polling place and using a three- or four-sheet ballot.

Printing ballots in advance, however, increases reliability and permits voters to daub their ballots even if all electronic equipment fails on election day. A three-sheet ballot would enable even greater distribution of printer trust but complicate a system already considered by some to be moderately complex.

Punchscan IBS exploits Punchscan's two-sheet ballot to permit distributing trust among multiple printers more easily than in traditional Punchscan. More field testing is required to gauge how well voters and election officials will handle this high-integrity voting system.

Chapter 6

CONCLUSION

Punchscan provides many desireable properties for a voting system. However, like any system, it makes certain assumptions that may require a separate system or security measures to guarantee. We have outlined these situations and provided novel solutions to these assumptions in the Punchscan system. Our final discussion summarizes these contributions and discusses some open problems.

6.1 Summary

This document has established that Punchscan makes two key assumptions that need to be addressed to ensure adequate system privacy. Most unrealisticly, it does not address the security around printing and transporting ballots to polling sites. It simply assumes that these things can be done securely, and we have explained how they can be done through various procedural protections and by expanding Punchscan to support them. Punchscan also assumes that election trustees will use a workstation with unique requirements, requiring us to design a novel new system to support it's requirements. These ideas are more useful if they can be used outside of the Punchscan system.

6.2 Discussion

The trusted workstation provides two simple yet interesting ideas stemming from the unique requirements of the Punchscan workstation. The software validation routine presents an acceptable way to validate software on benign hardware. It is hard to imagine where else such a validation system might be used, but we think it could be useful in digitized gambling applications like a poker dealer. Currently, players must trust such a dealer to fairly deal out cards, but the software validation routine might make such a system more acceptable to players and avoid the possibility of a fraudulent dealer.

UCSS is unique in that it is, as far as we know, the first system to do secret sharing without a dealer. Inherently, such a system requires a trusted workstation, because something must protect the participants from seeing the generated secret key. This requirement is fundamentally different from traditional secret sharing, where the key is never seen by the participants until it has been computed in a group computation. We predict that it could be useful for seeding a secure random number generator, or for other systems like Punchscan.

As far as we can tell, the IBS modification of Punchscan, in an of itself, is not particularly applicable to a situation besides voting. However, it does make Punchscan simpler, and provides a deeper understanding of how and why Punchscan works.

6.3 Open Problems

For all that Punchscan provides, it ignores Registration issues which are a critical part of any voting system. Without an accurate registration system to authenticate legitimate voters, election officials cannot prevent multiple voting, ballot stuffing, and other problems. While any registration system could be used, the protections in Punchscan could facilitate a registration system with much stronger properties. For example, it would be useful, if a mistake is found, to have the ability to take votes out of the final tally.

It is not clear that Punchscan, or any system, could ever fully avoid the problems it encounters with ballot printing. This problem is fundamental to cryptography. Each voter needs to receive the information necessary to hide his or her choices. While using a public key approach to encrypt votes can avoid this problem, it would be very useful if this could be accomplished without the help of a machine by using a paper ballot like the one that Punchscan provides.

The ballot format is unique to Punchscan. It may be too hard for voters to understand and effectively use. While the format can be changed, this also changes the security properties and protections of the system. More study is needed to say if this new ballot format is beneficial to voters.

6.4 Final Thoughts

The acceptance and use of Punchscan, or any system like it, depends on the security, usability, and completeness of the system. Punchscan still needs some usability improvements, but it offers advantages that seem impossible to many people.

With an improved user interface and a registration system Punchscan would be an ideal voting system. Punchscan is laying the foundation for a very bright and promising future, and it will not be long before it or a system based on similar ideas becomes popular.

Appendix A

HOW TO BUILD A CUSTOM LINUX LIVECD

N.B. Most of this information is a compilation of [1], [2], and [6].

1. Mount/Load the Kubuntu 6.06 LiveCD.

2. Copy the ”casper” and ”preseed” directories to your working directory.

```
sudo -s
mkdir livecd
cp -r /media/cdrom/casper livecd/
cp -r /media/cdrom/preseed livecd/
cp /media/cdrom/md5sum.txt livecd/
```

3. Create a 2 gigabyte ext2 filesystem image and mount it.

```
sudo dd if=/dev/zero of=ubuntu-fs.ext2 bs=1M count=2147
sudo mke2fs ubuntu-fs.ext2
mkdir ubuntu-fs
mount -o loop ubuntu-fs.ext2 ubuntu-fs
```

4. Mount the squashfs filesystem:

    mkdir squash-fs

    mount -o loop,ro livecd/casper/filesystem.squashfs squash-fs

5. Copy the data from the compressed system to the 2 gig partition and unmount the squashfs filesystem.

    cp -a squash-fs/. ubuntu-fs/

    umount squash-fs

6. Chroot to the directory, remove all unnecessary things (openoffice, games, etc).

    cp /etc/resolv.conf ubuntu-fs/etc/

    cp /etc/host.conf ubuntu-fs/etc/

    mount -t proc -o bind /proc ubuntu-fs/proc

    chroot ubuntu-fs /bin/bash

7. Note: steps involving ubuntu package management are not being covered here. Also, be sure to include the JRE, the PEW, and edit the JRE Security such that the PEW will run (you may also want to check this by trying to run a meeting.) When finished, cleanup.

    rm ubuntu-fs/etc/resolve.conf

    rm ubuntu-fs/etc/host.conf

    dd if=/dev/zero of=ubuntu-fs/tmp/tmpfile

    rm ubuntu-fs/tmp/tmpfile

umount ubuntu-fs/proc

8. Update the filesystem manifest.

   chroot ubuntu-fs dpkg-query -W –showformat='${Package} ${Version}\n'
   \
   > livecd/casper/filesystem.manifest

9. Create the new squashfs, and update md5sums.

   mv livecd/casper/filesystem.squashfs ./filesystem.squashfs.bak
   mksquashfs ubuntu-fs livecd/casper/filesystem.squashfs
   umount ubuntu-fs
   cd livecd
   find . -type f -print0 —xargs -0 md5sum — tee md5sum.txt

10. Mount the initrd to get the LiveCD scripts to use your USB Drive.

    mkdir initrd.dir
    cp livecd/casper/initrd.gz .
    gunzip initrd.gz
    mount -o loop,rw initrd initrd.dir

11. Edit the file initrd.dir/scripts/casper to go from this:

        case $fstype in
    vfat—iso9660—udf)

To this:

```
    case $fstype in
vfat—iso9660—udf—ext2—ext3)
```

12. Close and Save the initrd.

```
umount initrd.dir
gzip -n9 initrd
mv livecd/casper/initrd.gz ./initrd.gz.bak
cp initrd.gz livecd/casper/
```

13. Shred, zero out, and partition your USB drive with enough space on the first partition to fit what you've created.

```
shred -n 1 -z -v /dev/sdb
dd if=/dev/zero of=/dev/sdb
fdisk /dev/sdb
mkfs.ext2 /dev/sdb1
mkfs.ext2 /dev/sdb2
...
```

14. Mount your drive and copy the livecd data over to it.

```
mount /dev/sdb1 /media/usbdisk
cp -a livecd/* /media/usbdisk
```

15. Install the bootloader to the usb drive.

grub-install –recheck –root-directory=/media/usbdisk /dev/sdb

16. Create /media/usbdisk/boot/grub/menu.lst to look like the following.

    title Punchscan Engine Wrapper LiveUSB

    root (hd0,0)

    kernel /casper/vmlinuz boot=casper ramdisk_size=1048576 root=/dev/ram

    rw –

    initrd /casper/initrd.gz

    boot

17. Unmount the drive, and see if it boots. Copy the image to the other drives if it works without issues.

    dd if=/dev/sdb of=/dev/sdc

    ...

REFERENCES

[1] Customizing a (K)Ubuntu 6.04 Linux Live CD. http://www.atworkonline.it/~bibe/ubuntu/custom-livecd.htm, November 2006.

[2] Howto Install a Debian GNU/Linux system onto a USB flash thumbdrive. http://feraga.com/book/export/html/25, November 2006.

[3] HP-UX Secure Shell Installation Directions. http://docs.hp.com/en/T1471-90028/ch02s02.html, November 2006.

[4] Preventing Trojan Downloads. http://linsec.ca/syshardening/trojandownloads.php, November 2006.

[5] Punchscan Website. http://www.punchscan.org/, November 2006.

[6] Ubuntu Forum post on getting the LiveCD to boot on a USB drive. http://www.ubuntuforums.org/showpost.php?p=1221276&postcount=153, November 2006.

[7] Using MD5 Checksums to Verify Open Office Software. http://www.openoffice.org/dev_docs/using_md5sums.html, November 2006.

[8] BLAKLEY, G. Safeguarding cryptographic keys. *Proceedings of the National Computer Conference 48* (June 1979), 313–317.

[9] BOWEN, D. California Secretary of State Top to Bottom Review. http://www.sos.ca.gov/elections/elections_vsr.htm, August 2007.

[10] BRASSARD, G., CHAUM, D. L., AND CREPEAU, C. Minimum Disclosure Proofs of Knowledge. *Journal of Computer and System Sciences 37* (1988), 156–189.

[11] CHAUM, D. Secret-ballot systems with voter-verifiable integrity. United States Patent and Trademark Office, 7,210,617, January 2003.

[12] CHAUM, D., VAN DE GRAAF, J., RYAN, P. Y. A., AND VORA, P. L. High Integrity Elections. Cryptology ePrint Archive, Report 2007/270, 2007. `http://eprint.iacr.org/`.

[13] CHAUM, D. L. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM 24*, 2 (1981), 84–90.

[14] CHAUM, D. L. Blind signatures for untraceable payments. *Advances in Cryptology* (1983), 199–203.

[15] CHAUM, D. L. Secret-Ballot Receipts: True Voter-Verifiable Elections. *IEEE Security and Privacy 02*, 1 (2004), 38–47.

[16] CHAUM, D. L., CRÉPEAU, C., AND DAMGARD, I. Multiparty unconditionally secure protocols. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing* (New York, NY, USA, 1988), ACM Press, pp. 11–19.

[17] CHAUM, D. L., RYAN, P. Y., AND SCHNEIDER, S. A. A Practical, Voter-verifiable, Election Scheme. Technical Report Series CS-TR-880, University of Newcastle Upon Tyne, School of Computer Science, December 2004.

[18] CHOR, B., GOLDWASSER, S., MICALI, S., AND AWERBUCH, B. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract). In *26th Annual Symposium on Foundations of Computer Science* (Portland, Oregon, 21–23 1985), IEEE, pp. 383–395.

[19] CNN.COM. Butterfly ballot cost Gore White House. `http://archives.cnn.com/2001/ALLPOLITICS/03/11/palmbeach.recount/`, March 2001.

[20] COMMISSION, U. S. E. A. Voluntary Voting System Guidelines. http://eac.gov/vvsg_intro.htm, December 2005.

[21] ELGAMAL, T. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory 31*, 4 (1985), 469–472.

[22] ESSEX, A., CLARK, J., CARBACK, R., AND POPOVENIUC, S. The Punchscan voting system: VoComp competition submission. In *Proceedings of the First University Voting Systems Competition (VoComp)* (2007).

[23] FELDMAN, A. J., HALDERMAN, J. A., AND FELTEN, E. W. Security Analysis of the Diebold AccuVote-TS Voting Machine. http://itpolicy.princeton.edu/voting/ts-paper.pdf, September 2006.

[24] FISHER, K. Punchscan: Security Analysis of a High Integrity Voting System. Master's thesis, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, December 2006.

[25] FISHER, K., CARBACK, R. T., AND SHERMAN, A. T. Punchscan: Introduction and System Definition of a High-Integrity Election System. In *Preproceedings of the 2006 IAVoSS Workshop on Trustworthy Elections (WOTE 2006)* (Robinson College, Cambridge, United Kingdom, 2006), International Association for Voting System Sciences.

[26] FUJIOKA, A., OKAMOTO, T., AND OHTA, K. A Practical Secret Voting Scheme for Large Scale Elections. In *ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques* (London, UK, 1993), Springer-Verlag, pp. 244–251.

[27] FUND, J. How to Steal an Election. *City Journal 14*, 4 (Autumn 2004).

[28] GARDNER, R., YASINSAC, A., BISHOP, M., KOHNO, T., HARTLEY, Z., KERSKI, J., GAINEY, D., WALEGA, R., HOLLANDER, E., AND GERKE, M. Software Review and Security Analysis of the Diebold Voting Machine Software. Tech. rep., Security and Assurance in Information Technology Laboratory, Florida State University, Tallahassee, Florida, July 2007.

[29] HOSP, B., AND VORA, P. L. An Information-Theoretic Model of Voting Systems. In *Preproceedings of the 2006 IAVoSS Workshop on Trustworthy Elections (WOTE 2006)* (Robinson College, Cambridge, United Kingdom, 2006), International Association for Voting System Sciences.

[30] JAKOBSSON, M., JUELS, A., AND RIVEST, R. Making Mix Nets Robust For Electronic Voting By Randomized Partial Checking. In *Proceedings of the 11th USENIX Security Symposium* (San Francisco, CA, USA, 2002), Usenix Assoc., pp. 339–353.

[31] JONES, D. W. Chad – from waste product to headline. `http://www.cs.uiowa.edu/~jones/cards/chad.html`, January 2002.

[32] KARLOF, C., SASTRY, N., AND WAGNER, D. Cryptographic Voting Protocols: A Systems Perspective. In *Proceedings of the 14th USENIX Security Symposium* (August 2005).

[33] KENT, A. Unconditionally Secure Bit Commitment. *Physical Review Letters 83*, 7 (August 1999), 1447–1450.

[34] KIM, G. H., AND SPAFFORD, E. H. The design and implementation of tripwire: a file system integrity checker. In *CCS '94: Proceedings of the 2nd ACM Conference on Computer and communications security* (New York, NY, USA, 1994), ACM Press, pp. 18–29.

[35] KOHNO, T., STUBBLEFIELD, A., RUBIN, A. D., AND WALLACH, D. S. Analysis of an Electronic Voting System, May 2004.

[36] LERER, L. Whose Polls Are Problematic? *Forbes.com* (November 2006).

[37] MENEZES, A. J., VAN OORSCHOT, P. C., AND VANSTONE, S. A. *Handbook of Applied Cryptography*. No. 0-8493-8523-7. CRC Press, 1996.

[38] NAOR, M., AND SHAMIR, A. Visual Cryptography. *Lecture Notes in Computer Science 950* (1995), 1–12.

[39] NECHVATAL, J. *Public-Key Cryptography*. National Computer Systems Lab, Gaithersburg, Maryland, April 1991.

[40] NEFF, C. A. Practical high certainty intent verification for encrypted votes., October 2004.

[41] NEFF, C. A. Verifiable mixing (shuffling) of El-Gamal pairs., October 2004.

[42] NEFF, C. A., AND ADLER, J. Verifiable e-voting. Tech. rep., VoteHere, 2003.

[43] OKAMOTO, T. Receipt-Free Electronic Voting Schemes for Large Scale Elections. In *Proceedings of the 5th International Workshop on Security Protocols* (London, UK, 1998), Springer-Verlag, pp. 25–35.

[44] PAILLIER, P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT* (1999), pp. 223–238.

[45] POPOVENIUC, S., AND HOSP, B. An Introduction to Punchscan. In *Preproceedings of the 2006 IAVoSS Workshop on Trustworthy Elections* (Robinson College, Cambridge, United Kingdom, 2006), International Association for Voting System Sciences.

[46] RABA INNOVATIVE SOLUTION CELL, R. Trusted Agent Report Diebold AccuVote-TS Voting System. http://www.raba.com/press/TA_Report_AccuVote.pdf, January 2004.

[47] RUBIN, A. Trusted distribution of software over the Internet. *Symposium on Network and Distributed System Security 00* (1995), 47.

[48] SALTMAN, R. G. *The History and Politics of Voting Technology: In Quest of Integrity and Public Confidence*. Palgrave Macmillan, January 2006.

[49] SCIENCE APPLICATIONS INTERNATIONAL CORPORATION, S. Risk Assessment Report Diebold AccuVote-TS Voting System and Process. http://www.verifiedvoting.org/downloads/votingsystemreportfinal.pdf, September 2003.

[50] SHAMIR, A. How to share a secret. *Communications of the ACM 22*, 11 (1979), 612–613.

[51] SHERMAN, A. T., GANGOPADHYAY, A., HOLDEN, S. H., KARABATIS, G., KORU, A. G., LAW, C. M., NORRIS, D. F., PINKSTON, J., SEARS, A., AND ZHANG, D. An Examination of Vote Verification Technologies: Findings and Experiences from the Maryland Study, April 2006.

[52] SHERMAN, A. T., GANGOPADHYAY, A., HOLDEN, S. H., KARABATIS, G., KORU, A. G., LAW, C. M., NORRIS, D. F., PINKSTON, J., SEARS, A., AND ZHANG, D. An examination of vote verification technologies: Findings and experiences from the maryland study, April 2006.

[53] STINSON, D. R. *Cryptography Theory and Practice*, 3 ed. Chapmand & Hall/CRC, Boca Raton, FL 33487-2742, 2006.