"Bulletproof Electronic Voting" —*IEEE Spectrum*

The Punchscan Voting System

VOCOMP Competition Submission

Aleks Essex and Jeremy Clark
School of Information Technology and Engineering (SITE)
University of Ottawa
Ottawa, Ontario
{aesse083,jclar037}@site.uottawa.ca

Richard T. Carback III
Cyber Defense Lab
Center for Information Security and Assurance
University of Maryland, Baltimore County
Baltimore, Maryland
carback1@umbc.edu

Stefan Popoveniuc
Department of Computer Science
George Washington University
Washington, D.C.
poste@gwu.edu

Executive Summary

Punchscan is an end-to-end independently verifiable cryptographic voting system, which was originally proposed by David Chaum in December of 2005. We, a team of students from the University of Ottawa, George Washington University, and the University of Maryland, Baltimore County have designed, developed, implemented and tested Punchscan in a binding election hosted by the University of Ottawa in March of 2007. In this document we will detail this effort.

At pennies a ballot, Punchscan is low-cost, open source, and can be run on commodity hardware. Unlike conventional electronic voting systems, the judicious use of cryptography alleviates the concern over the strict use of trusted hardware and software components. The polling place hardware/software never encounters a voter's "unencrypted" vote. The entire process leaves an end-to-end audit trail that allows voters to independently verify for themselves that their ballot was correctly included in the tally, and that the tally was performed correctly. Additionally by design, audits can be performed throughout the election making Punchscan highly effective at preventing deliberate cheating or software errors that might necessitate a re-vote.

Punchscan provides transparency to the entire election process: mandatory public pre and post election public audits, and the ability for a voter to check the correct printing and recorded marks on a paper receipt she keeps. Even though a voter can verify her vote were counted as cast, the ballot receipt does not contain enough information to show someone else how she voted.

Unlike other cryptographic voting systems, you don't need a degree in abstract algebra to understand how Punchscan works. The cryptographic clockwork that protects, shuffles and retrieves the ballots, called the "Punchboard," is similar in technique as the "secret decoder ring" one might encounter in a cereal box. And because Punchscan does not lean on trusted software it is easier than convincing anyone how software in current voting systems works-even if we were allowed to see it!

These unique properties produce a system with a voluntary and universally available process that establishes an overwhelmingly high statistical degree of confidence in the integrity of the outcome. These ideas are new and have the potential to radically change the way we think about and build the voting systems of the future.

Contents

Chapter 1

Introduction

Punchscan is an open-source voting system, the results of which are verifiable by voters through their participation in the election audit process, a voluntary and universally available process that establishes an overwhelmingly high statistical degree of confidence in the integrity of the outcome. Engaging voters more centrally in the election process through independent verification, observability and transparency is the motivation behind the development of the Punchscan voting system. At the same time this system was developed around the recognition that the secret-ballot, and in turn voter privacy, remains a fundamental requirement in modern voting systems. Engineering a voting system that offers voters the ability to "see their vote count," while at the same time protecting against improper influence is an enormous design challenge, representing the current state-of-the art in the field of voting systems research. The ultimate goal of this effort is a voting process that preserves and enriches the quality of democracy in this and other countries around the world. We are submitting Punchscan to the Vocomp student voting competition as major advancement toward this goal.

1.1 Independent Verification

The independent verification process is Punchscan's raison d'être. This notion of independent verification refers to an individual or organization (unassociated with the election trustees) that undertakes to perform an audit (i.e. verify the correctness) of election results. This represents a new paradigm in elections brining citizen oversight to a new level. Unlike conventional optical scan voting systems, the Punchscan audit is:

- End-to-end, performed on all ballots (not just a small percentage).
- Compulsory, performed every time (not only under exceptional circumstances).
- Available to and repeatable by anyone.
- Open specification, open source
- Software/hardware independent
- Easy enough to perform that it encourages widespread participation

Independent verification is at the heart of the Punchscan system. The results of a Punchscan election are verifiable by voters through their participation in the audit process, a process that establishes a high statistical degree of confidence in the integrity of the outcome. When the election authority prints the ballots prior to an election, the unique information contained in each ballot is "committed to" using a cryptographic one-way function. Though this commitment is made public, the actual information contained on the ballot remains sealed. Because the function is one-way, it is computationally infeasible to determine the information on the sealed ballot given only its publically posted commitment. In Punchscan, the ballot receipt represents an "encryption" of the voter's vote. These ballot receipts (aka "encrypted ballots") will eventually be run

through a decryption operation which produces "plaintext ballots," (aka results) containing voters' votes. These resultant ballots are shuffled, and not directly linked to their original serial numbers.

Independent verification however comes at a price; ballots must contain unique information. Because Punchscan adheres to the traditional principle of the secret ballot, the system must contain additional privacy enabling and assurance components. Therefore as we begin examining the Punchscan system, keep in mind that we are attempting to design a voting system with the following two properties:

- **Integrity** of election results – through ballot "receipts" and independent verification (i.e. audit).
- **Secrecy** of voters' vote – through strong cryptographic design and distribution of trust.

Intuitively one might expect that integrity of election results and ballot secrecy are mutually exclusive. However as we will see, through the use of cryptography and careful design, these two properties are both simultaneously realizable.

1.2 E2E: End-to-End Cryptographic Independent Verification Voting Systems

In 2005, the American Election Assistance Commission (EAC) released a set of voluntary voting system guidelines [1] that includes a description of what they refer to as "End to End Cryptographic Independent Verification" (E2E) systems. According to the EAC, typical distinguishing features of an E2E voting system are as follows:

- A paper receipt is issued to the voter that contains information that permits the voter to verify that her choices were recorded correctly. The information does not permit the voter to reveal her selections to a third party.
- The voter has the option to check that her ballot selections were included in the election count, e.g., by checking a web site of values that should match the information on the voter's paper receipt.
- Such a system may provide an assurance not only that her ballot choices were correctly recorded (cast-as-intended), but that those selections were included in the election count (counted-as-cast).

1.3 Punchscan in the scope of E2E

Punchscan has been designed to be an E2E voting system. Most of the system components you will read about in the following chapters have been developed to realize the E2E criteria. The EAC found that the range of proposed E2E systems have points of commonality, and they attempted to summarize these in a list of properties. Here we present some of them and briefly explain how Punchscan does or does not exhibit their properties. Note that this is not a list of requirements for a system to be classified as E2E but rather a preliminary sketch of the typical properties of these systems.

Property 1. *Voters' ballot selections are encrypted for later counting by designated trustees.*
A Punchscan ballot consists of two pages. The top page contains a list of contests and candidates with set of randomly ordered symbols beside the candidate names. There are holes in the top sheet that display a corresponding (but independently and randomly ordered) set of symbols. To vote on a Punchscan ballot, the voter observes the symbol appearing beside their chosen candidate's name, and locates the matching symbol in the holes. The voter then marks that hole with an implement such as a bingo-style dauber. The implement is sized slightly larger than the hole such that the ink mark will be made on both sheets. One of these sheets is destroyed in a cross-cut paper shredder. The remaining sheet represents the voter's receipt and is now "encrypted." [1] Only the threshold number of trustees (aka the election authority) have the ability

[1] Since both sheets contain random but independent orderings of the symbols, possessing only one of the sheets does not give you information about the corresponding symbol on the other sheet.

to reconstruct the information contained on the destroyed sheet.

Property 2. *Voting will produce a receipt that would enable the voter to verify that their ballot selections were recorded correctly and counted in the election.*
Punchscan uses a robust audit procedure, including a process by which a voter can visit the election website and look up their ballot using the serial number contained on their receipt and verify what they hold in their hand matches what was recorded by election authority.

Property 3. *The receipt preserves voter privacy by not containing any information that can be used to show the voter's selections.*
Because one of the sheets is shredded, and assuming that the ordering of symbols contained on that page were uniformly random and independent from the page that was retained (aka the receipt), then no information about the destroyed sheet is contained on the retained sheet, and therefore the vote cannot be guessed with any advantage.

Property 4. *No one designated trustee is able to decrypt the records; decryption of the records is performed by a process that involves multiple designated trustees.*
Punchscan employs a threshold based password scheme whereby a pre-designated number of trustees must correctly enter their passwords before the records can be reconstructed.

Property 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.*
Punchscan in its original form relies on voter receipts to reconstruct an election should the counted records be destroyed. However, as will be discussed in the next section, the implementation of Punchscan used in this case study expanded the originally proposed system to include a paper-based backup of the ballot receipts.

Property 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.*
The backup ballot receipts used in this election contained a serial number which matched the serial number of the ballot. While the ballot receipts themselves were digitally signed, the paper backups were not as it was agreed that the backup records were very unlikely to be needed. Should they have been used, they would have been published and thus their integrity would be ensured through voter verification. In future elections, consideration will be given to digitally signing the backups as well as the receipts.

Property 7. *The documentation includes extensive discussion of how cryptographic keys are to be generated, distributed, managed, used, certified, and destroyed.*
The source code for all the software used by Punchscan is open source and can be examined by anyone. Furthermore, the Punchscan team has attempted to document the underlying cryptography of the system through papers, presentations, and other documentation available from the Punchscan website [2].

Property 8. *Vote capture stations used in end to end systems must meet all the security, usability, and accessibility requirements.*
The security of the vote capture station in a Punchscan election is similar to that of a paper ballot voting station. Two additional security measures are taken: one is to lock the ballot to a clipboard and the second is to ensure a high-integrity paper shredder. The purpose of this case study, in part, is to examine the usability of Punchscan and will be discussed further in later sections.

Property 9. *Reliability, usability, and accessibility requirements for printers in other voting systems apply as well to receipt printers used in end to end systems.*
Punchscan can be easily implemented with inexpensive off-the-shelf equipment. As will be examined further

[2] http://punchscan.org

in this case study, reliability and usability issues emerged. However Punchscan is largely hardware independent and could be adapted to use proprietary voting-dedicated equipment.

Property 10. *Systems for verifying that voter ballot selections were recorded properly and counted are implemented in a robust secure manner.*
Punchscan allows the voter to verify the proper scanning of their ballot at the polling station before it is cast, in addition to their ability to check the receipt online. The security of the Punchscan tallying process is dependent on well-studied cryptographic primitives [16] and no implementation vulnerabilities have been discovered to date.

Chapter 2

System Description

The overarching idea that enables independent verification in Punchscan (and many E2E systems) is the cryptographic principle of "cut and choose" (CNC) [5]. 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 successfully cheat without being caught, Alice creates 1 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 to simultaneously 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.

Our description of Punchscan begins with at a high-level design view, giving an overview of the core components and the auditing protocol. Afterward, we give a more detailed architectural view, describing the physical hardware and software components, user roles, and how they work together in the system. These sections are derived with permission from the published work done in [16, 9].

2.1 Core Components

At its core, Punchscan is a derivative of an earlier unnamed E2E system proposed by Chaum that used visual cryptography [4]. Like that system, Punchscan also uses a two-sheet ballot and an audited mixing of ballots, but both of these components are substantially less complex in Punchscan.

The key advantage of Punchscan over that system 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 it does not use visual cryptography. We believe 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 a voter's vote.

Instead of a series of tellers performing mix operations, Punchscan relies on a simple auditable mix network based on two cryptographic primitives: a cryptographically secure pseudo random number generator (CSPRNG) [15] and an unconditionally secure bit commitment scheme (USBCS) [11]. 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*.

We discuss the trade offs and advantages of this and other choices in Chapter 9. Now, we discuss the ballot, our mix network 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.

2.1.1 Ballot

The Punchscan ballot, as illustrated in Figure 2.1 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.

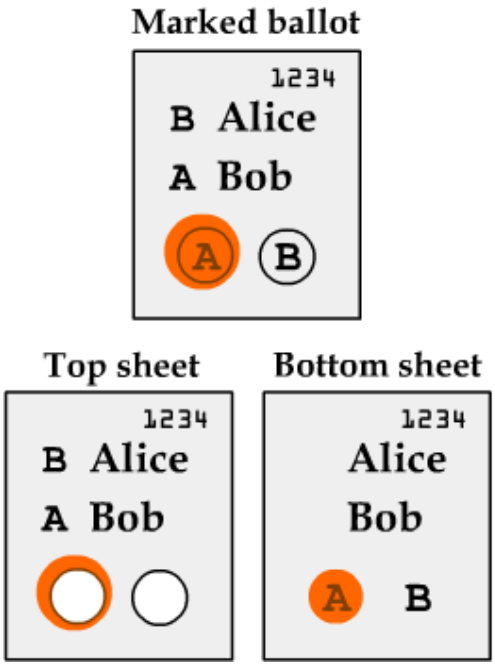


Figure 2.1: A simple Punchscan ballot showing candidate list, serial numbers and letters appearing on each respective sheet. The letters on each sheet are independently randomly ordered. This diagram denotes a vote for "Bob."

To vote, each voter uses a bingo dauber to mark the letter seen on the bottom sheet 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 much larger than the hole through which the letter is viewed. Afterward, either the top or the bottom sheet is destroyed (as randomly determined by a poll worker before the voter is issued her ballot), and the surviving sheet is scanned, publicly posted, and kept by the voter as a receipt[1]. As shown in Figure 2.1, neither half of the ballot can reveal the original vote by itself. Only the *Election Authority (EA)* can determine the original intent, and it does so using the Punchboard. 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.

[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. This change removes the need for a choice of sheet (forcing the bottom sheet to be the choice). We do not do this 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 to check the integrity of the printing process.

2.1.2 Punchboard

In order to determine voter intent, the trustees must know the letter ordering on the destroyed half of the ballot. This information is contained in a special data structure which we refer to as the "Punchboard," and example of which is shown in Figure 2.2. To represent votes or marks in the Punchboard by candidate order. Thus a 0 in one of the cells of the Results table represents a vote for the first candidate listed on the ballot, a 1 represents the second candidate and so on. We refer to this as the "canonical ordering;" the ordering of candidate names as it appears on the ballot.

Figure 2.2: The Punchboard demonstrating 8 ballots in the simplest election: a single contest, two-candidate plurality (i.e. "first past the post") vote. "Encrypted" votes are displayed on the left side in the P table and "unencrypted" votes are shown in the R table. The Flip columns 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.

The Punchboard is used to provide voter privacy and election integrity. If we post it as shown in the figure, 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. This method enforces integrity by making public certain values as we progress through the election, 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 protects the privacy of voters.

In Figure 2.3, 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 is revealed when a ballot is spoiled, or after results are posted when the EA 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. This data is released only when a ballot is spoiled. Revealing this data can check on both the printing process and serve as an integrity check to make sure the flip columns correspond to the top and bottom sheet 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 is an explicit CNC

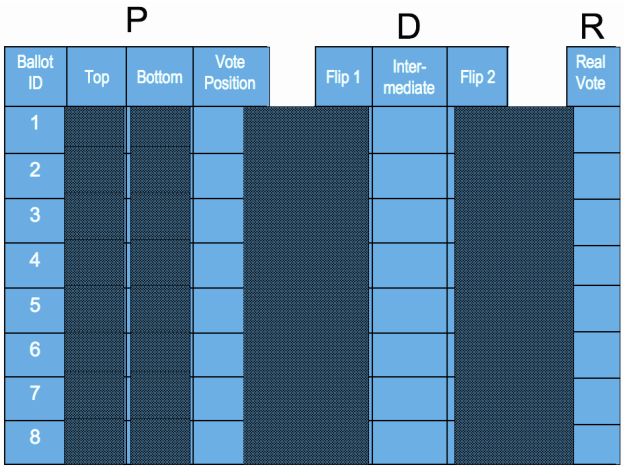


Figure 2.3: The punchboard as published before any auditing. Each cell in the P table is committed to using a USBCS. Each Flip column in the D table, and the rows in the P and R tables it corresponds too are also committed.

operation, and allows us to verify that the table was filled out correctly by the EA, but does not permit us to determine what rows or votes in the P table corresponded to what rows or votes in the R table.

2.2 Independent Verification in Punchscan

Let us reiterate that independent verification is the ultimate purpose of Punchscan and E2E systems in general. Punchscan realizes independent verification through four verification mechanisms:

1. Audit Challenge
2. Pre-Election Audit
3. Post-Election Audit
4. Online Receipt Check

The election trustees will "publish" (i.e. make publically available) selected pieces of information about the Punchboard that will be used for the purposes of independent verification. This published information does not allow anyone to link voters to votes, but does offer a high statistical assurance that the election results are legitimate.

2.2.1 Audit Challenge

We begin by discussing the notion of an audit challenge. To preserve ballot secrecy, we cannot open up the entire Punchboard to be audited. However the Punchboard has been modularly designed such that pieces of it can be revealed without compromising ballot secrecy. Which subset of the Punchboard is to be revealed is decided through an audit challenge. This could simply be candidates each requesting ballots to be audited, or even flipping a coin. However we have chosen to implement our audit challenge generator to be such that no one will know a-priori which selections will be made. The audit challenge generator serves the following purpose:

- **Pre-Election Challenge**: Given the total number of ballots in the election, randomly select serial numbers of half the ballots to be published (i.e., spoiled) and audited.
- **Post-Electon Challenge**: Given the serial numbers of the ballot receipts cast in the election, for a given ballot select either the left or right half of the decryption table to be published and audited.

If the election authority does not know *a priori* which ballots will be checked, it faces a high probability of getting caught if it publishes false commitments. The ballots to be selected for the audit should not be guessable with any advantage at the time the commitments are published. Additionally the challenges themselves must be verifiable after the challenges are made. On first blush it would seem we would require some form of random process (e.g. rolling dice) to perform these challenges. However unless you trust the dice and are personally available to witness it being rolled, you cannot be fully satisfied the process was not manipulated to ensure a particular outcome. In the following chapter we describe our implementation of the audit challenge generator as using stock indeces to produce fair and observable challenges.

2.2.2 Pre-Election Audit

The pre-election audit ensures proper construction of the Punchboard. During the pre-election audit, *half* the ballots generated by the trustees are selected to be examined. These ballots are unsealed (and spoiled) and checked to ensure they are properly formed (i.e., they will correctly perform the decryption) and that they match their commitments. Given a fair (i.e. randomly chosen) selection, we can be satisfied that the remaining sealed ballots are properly formed.

Figure 2.4: Pre-Election Audit

2.2.3 Post-Election Audit

During the post-election audit, for each ballot cast in the election, one half of its decryption transformation (either left or right) is unsealed, published, and checked to ensure it correctly performs its half of the decryption. The left partial-decryption step corresponds to the decryption of the ballot receipt to the partially decrypted result. Conversely the right partial-decryption step corresponds to decryption of the partially decrypted result to the result (i.e. vote). Again, given a fair (i.e. randomly chosen) selection, we can be satisfied that the remaining sealed half of the decryption is valid.

Figure 2.5: Post-Election Audit

2.2.4 Receipt Check

The receipt check is the process by which the voter verifies that the information contained on their paper ballot receipt matches the information that was used in decryption/tally. The receipt check needs to be easy and available to voters to promote their participation in the process. It is however not mandatory for the voter to perfom this step, and even a high degree of confidence in the election results can still be obtained with a relatively small number of checks. Additionally since the receipt does not contain information about how you voted, you can share this information with others, and allow them to perform this verification step on your behalf.

Therefore receipt information can be published freely, for example in a newspaper. We have chosen instead to implement the receipt check as an online solution. The voter visits the election website, types in the serial number appearing on their ballot receipt. A diagram of their ballot is displayed allowing the voter to verify that what they hold in their hand matches what was used in the decryption/tally.

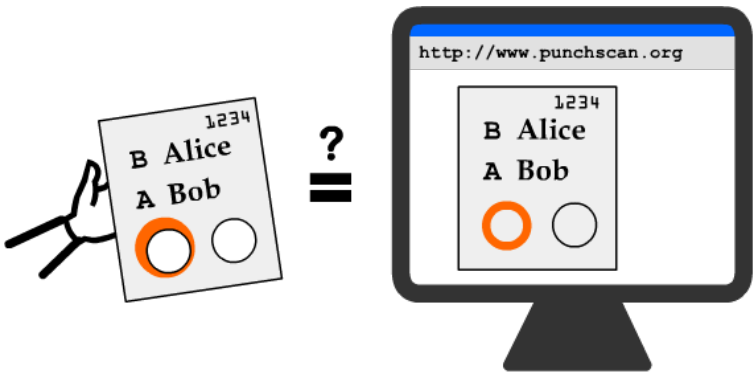


Figure 2.6: Online Receipt Check

2.3 System Components

This section will cover the components of the system that are not people and are necessary at an architectural level. Specifically, this does not include independent or unconnected things like the ballot clipboard that is discussed later.

2.3.1 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 EA responds 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 securely upload updated election data. 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.

2.3.2 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.

A simple USB key may serve as a removable and recordable storage medium. Any such device can adequately supply and store data, as long as it features a write-protect switch to optionally prohibit the deletion

or alteration of data or programs. Implementations may employ CD or DVD media and a combination of read-only and recordable disc drives to accomplish the same task.

Since the Diskless Workstation is the only computer to process election data in unencrypted form, a high threshold is set on its security and integrity. Its hardware configuration limits its ability to store or transmit sensitive information, and its Verified Trusted Software should faithfully process all data according to the descriptions earlier in the chapter.

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

2.3.3 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 trusted to print each ballot as directed by the Punchboard's Print table. Printing distribution strategies can be used to minimize the impact of a violation of that trust.

2.3.4 Scanner

Within the polling place, Voters mark their ballot and separate 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 an XML file 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 can be done using software algorithms or by calibrating the Scanner with a sample ballot with all positions marked.

2.3.5 Ballot Authoring

One final software program is needed to specify key ballot parameters. The EA uses 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.

2.4 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 its correctness. We now present and describe the four roles of Election Authority, Poll Worker, Voter, and Auditor.

2.4.1 Election Authority

The Election Authority (EA) which includes the election trustees is responsible for administrating and running the election. As a group they are trusted to handle all election data, including the Punchboard, in both its encrypted and unencrypted forms. It is essential that the election machinery distribute access to sensitive data across a super-majority of trustees, such no that trustee could ever view the unencrypted secret election data.

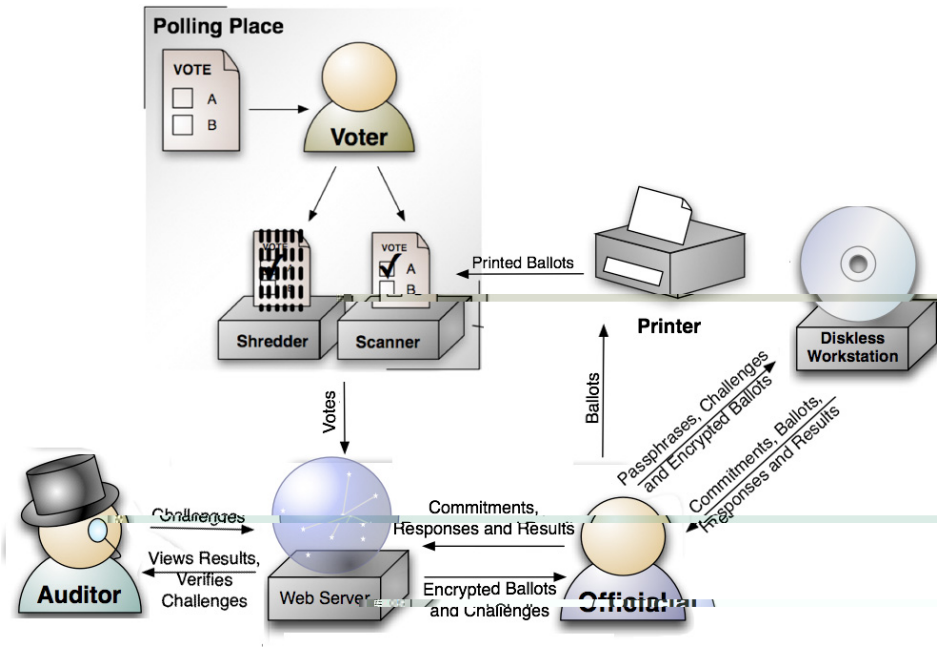


Figure 2.7: An architecture diagram of the Punchscan System. The web server acts as a central, transparent repository for all election data. Not shown on this diagram is "everyone"; anyone can check receipts or download election data from the Webserver.

2.4.2 Poll Worker

Poll Workers are the volunteers and other election officials that are responsible for the proper operation of each polling place. We think of the EA as actually a small group (no more than 5 or 10 people). Poll Workers themselves have various roles from manning tables, to passing out ballots, to directing what polling places printers send their ballots.

2.4.3 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.

2.4.4 Auditor

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

2.5 System Architecture

An illustration of our system architecture is shown in Figure 2.7[2]. As can be seen, the web server acts as a central repository for all election data, with the EA Officials responsible for populating the vast majority of that data (except for votes and audit challenges). Auditors are responsible for providing challenges (choosing

[2]This diagram was created from graphical elements that are found in [9]. They were created by Kevin Fisher.

random numbers, effectively). They must also verify the proper disclosure of data, however, anyone can do so if motivated. Voter's play a key role in generating the ballots, but what is not shown is that we rely on a small number to also verify their receipts with the web server. This is not indicated on the diagram because anyone can check ballots, and the web server should not make a distinction between an observer and a voter. We also see the printer and diskless workstation, the two components that are trusted with privacy and interacted with exclusively by the EA officials.

Chapter 3

Function Specifications

3.1 Ballot Template Design

The layout of a Punchscan ballot is created using ballot template software, which can be done using almost any standard desktop publishing application. For the student election, we used Microsoft Publisher. In addition to whatever standard text and graphics are placed on the ballot, the only requirement for the ballot template is the ability to mark certain locations with simple colored shapes. The specification calls for colored disk shapes to represent items that will appear on the top page, and colored ring shapes to represent items that will appear on the bottom page. These positional markers are used to place four categories of items on the ballot:

1. Ballot serial number (top and bottom pages).
2. Letters appearing beside candidate names (top page).
3. Letters appearing through the holes (bottom page).
4. Scanner alignment marks (top and bottom pages).

The template is output is a PDF file, and a sample template is shown in Figure 3.1. Punchscan also offers a template file for MS Word that uses macros and style profiles to automate the process of creating a ballot. MS Word can save the document to a PDF with a free extension available from Microsoft.

3.2 Ballot Authoring

Ballot authoring accepts a PDF file with special marks. These special marks are provided to the Ballot designer and pasted into the document.

1. Recognizes special marks on provided PDF (see Figure 3.1).
   (a) Donuts are bottom sheet data.
   (b) Disks are top sheet data.
   (c) Green marks are for serial numbers.
   (d) Yellow marks are for alignment marks (for both top and bottom sheet).
   (e) Blue, Orange, or Other are for character placement.
2. Output: XML Specification of ballot, Industry standard drill press files for drilling holes on paper, and PDFs for top and bottom sheets without special marks (except yellow alignment marks).

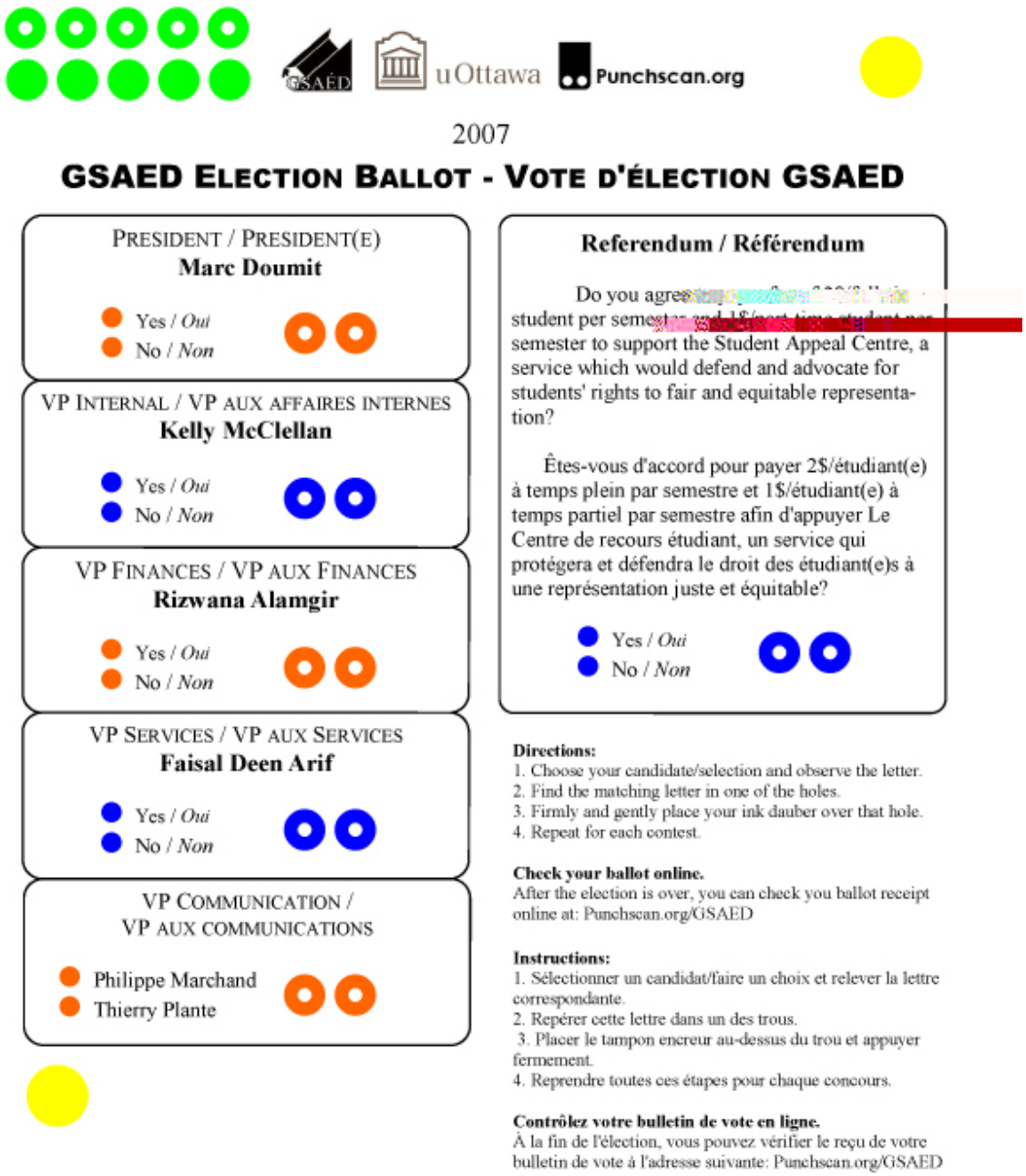


Figure 3.1: Ballot template used in the GSAÉD student election (described in chapter 6). Colored disk and donut shapes indicate the location where serial numbers and permutations will eventually be placed on the actual ballots.

3.3 Election Engine

The election engine forms the core of the audit and tally operations. It creates and populates the Punchboard, generates audit data, and decrypts and tallys the votes. Additionally, as part of the integrity of the system, it has a means by which the trustees can validate the software, and generate election master keys in a safe way.

3.3.1 Software Validation Tool

Before an election begins, an image of the trusted software should be posted online and available for download to be analyzed. During the election, this image is copied independently by members of the EA and brought to the meeting. The image itself will support a software validation function that simply hashes the disks it is told to hash. In order to perform the software validation EA members will perform the following actions:

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

If the hardware is benign, and unless all of the software is corrupt or the validation is compromised, this process ensures that any malicious or in correct software is discovered before any protected operations are performed or any sensitive data is entered into the system. It is still possible to sneak malicious software into the system through the bios or hidden modifications to the hardware, but we believe that this provides a cost-effective measure for protecting privacy guaranteed by proper operation of the system.

Unfortunately, this can be rather tedious and slow, especially if there are alot of EA members, but there are ways that it can be sped up. Members could agree on trusting a subset of members, or they could randomly pick members to run their software by rolling dice or some other method of picking random numbers.

3.3.2 Trustee Election Master Key Creation

Recall that one of the core E2E criteria is the ability to distribute the election master key among a plurality of election officials (aka trustees). Ideally these trustees form a zero-sum relationship meaning that they have strong incentive (personal, political or otherwise) to maintain the secrecy of their pass phrase. Furthermore it may also be desirable to be able to regenerate the master key (at subsequent meetings of the trustees) even in the even that some trustees are unable to be in attendance.

To achieve these properties, the Punchscan team has developed and implemented a user-contributed secret sharing (UCSS) scheme that relies only on standardized hash and symmetric cipher based cryptographic primitives. The UCSS step will produce or retrieve the public/private key pair as well as the election master key. The "share" that the trustees contribute is in the form of a username/password combination. All trustees must be present at the first meeting. Each subsequent meeting, if all the trustees are present, the software will rerun the key generation step. If the necessary subset is present, it will perform key retrieval (secret sharing). The software determines which scheme will be used based on how many participants enter their shares.

Entering Shares

Each of the trustees enter a username and password. If this is not the first meeting of the trustees, each username and password is checked against a password database. The password database is an encryption of a public constant with a key generated by a hash of their usernames and passwords concatenated together. If this check fails, an error is recorded in the audit log and that username and password 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 password concatenated together. Before moving on to key

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

Table 3.1: Format of UCSS passwd file. E is the encryption function, H is the hash function, and c is the public constant.

generation, the software will save this file in the Private storage directory chosen by the participants in the format is shown in table 3.1.

Key Generation and Retrieval

If all participants were present and their usernames and passwords are correct or this is the first time the group has met, then we perform key generation as follows: All usernames are concatenated together, and all passwords are concatenated together, then the concatenation of usernames and passwords 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 randomly generate a Public/Private key pair. As an example, for five users $u_1, ..., u_5$ with passwords $p_1, ..., p_5$ and the hash function H, the secret key S is:

$$S = H(u_1u_2u_3u_4u_5, p_1p_2p_3p_4p_5) \tag{3.1}$$

To store the keys, the software needs a threshold 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 passwords, 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 password 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 passwords $p_1, ..., p_3$, 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:

$$E_{H(u_1u_2,p_1p_2)}(S) \tag{3.2}$$
$$E_{H(u_1u_3,p_1p_3)}(S) \tag{3.3}$$
$$E_{H(u_2u_3,p_2p_3)}(S) \tag{3.4}$$

This scheme is far from space efficient requiring n choose $n - k$ 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 their own share and contributes to the key generation process.
4. The keys are regenerateable if the drive containing the encrypted password 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 password.

3.3.3 Response to Audit Challenges and Ballot Decryption

Audit challenges are discussed in great detail in section 3.8. What the engine does at this point depends on what meeting the EA is running, which is described in section 3.7.2. To post the results, it will publish the data associated with the Intermediate and Results columns from the Punchboard in an XML format. The engine does not tally data. Output data is stored on a specific USB drive or other Media, then taken to a separate computer and uploaded to the Webserver.

3.4 Polling Place Software

Polling place software will scan ballots and store them in as digital images and a representation in a standard XML file.

3.4.1 Initialization

Poll workers start the polling place software, enter authorization data (which may be pregenerated or initialized, preferably pregenerated), scans a fully marked ballot (one in which all possible mark positions are marked), and selects a mark color to calibrate the ballot by indicating alignment mark and dauber colors.

3.4.2 Optical Scan Recognition

The optical scanning will periodically check a specified location where ballot images from the scanner are stored. It then performs the following tasks:

1. Find alignment marks, and adjust image with standard 3D transformations. This will Zoom, center vertically/horizontally, and rotate the image according to the BallotMap file created by the Ballot Authoring tool.
2. Determine top/bottom sheet.
3. Interpret serial number. This posts what ID number the scanner thinks it saw on the screen in a designated place.
4. Find marks on the receipt.
5. Interpret marks, putting checks over legal marks and X's over illegal marks. This is to prevent overvoting and other problems. The information to do it is found in the BallotMap file.
6. Generate an XML representation of the scanned receipt with the above information. Show a "Cast Ballot" button if the ballot has no illegal markings.

3.4.3 Overlay Printouts

This functionality initializes when a voter indicates to the scanning software that her ballot should be cast by pressing the "Cast Ballot" button. It will instruct a printer to print data.

- Input: XML Representation of the receipt.
- Output: Printer data to mark unmarked positions and marked positions as interpreted by the scanner (X's and O's), and a barcode of a digital signature of what was seen on the receipt using poll worker credentials.

3.5 Election Webserver

The Election Webserver will store all public election data. This includes the Punchboard data in an XML format the Auditing software can understand, and receipts that voter's can check. The server also stores news events and other information for voters, as well as auditing software for voters who are interested to check the data themselves.

3.5.1 Voter Database

The voter database was a requirement of GSAÉD and not part of the Punchscan system (which does not have a registration component, yet). It is included for completeness, as we did use it in the GSAÉD election. It allows Poll Workers to verify voter identity and mark that voters have voted.

- Initialization - Accept voter ID numbers and names, generate database
- Input: Voter ID number
- Output: Voter marked as voted and success message if exists and not already marked voted, or error message.

3.5.2 Meeting Upload

Permits EA to upload all meeting data (the Punchboard) to standard storage location. This can be done manually, but was facilitated by a script. It simply takes the file as input, and adds it to the correct folder. Later, it is linked to by the webmaster.

3.5.3 Receipt Upload

Permits scanning locations to upload receipt data with appropriate authorization credentials.

- Input: XML Specification of ballot marks, ID, and sheet (top or bottom). This can be 1 or more ballots.
- Output: XML Data stored in database system for later retrieval.

3.5.4 Receipt Checker

Allows voters to check receipt data. See the following section for more information about this step.

- Input: Receipt ID number
- Output: A computer generated image of the receipt from database information, or error image if that receipt does not exist.

3.5.5 Bulk Receipt Download

- Input: None
- Output: All stored ballot receipt IDs, marks, and sheet (top or bottom) in XML format. This is identical to the scanning place XML Specification.

3.5.6 Results

- Input: XML Meeting file with results data.
- Output: A summary image with result totals next to candidate names.

3.6 Cryptographic Specification

3.6.1 Ballot Keys

Each individual ballot requires its own "key" which will later be used by the commitment function to "commit to" (encrypt/hash) the secret information contained in the ballot. Although these commitments are made public, the actual information contained in the ballot remains sealed. To create these ballot "keys," we use some sort of unique identifier of to function as "key material." We denote the key material of ballot i as (Km_i). The election master keys Mk_1 and Mk_2 are generated by the trustees during their first meeting. Typically the ballot serial number can fulfill this function. There is also an election constant C which protects the election master keys across different elections, should trustee passwords ever reused for whatever reason.

Algorithm 1: Ballot Key

Input: Km_i, Mk_1, Mk_2, C
Output: Skm
1 $\text{Skm}_i = D_{Mk_1}(C \oplus E_{Mk_2}(C \oplus E_{Mk_1}(Km_i)))$

In our implementation, the encryption/decryption functions listed above are performed using AES with a 128-bit key size. The ballot key generated from the key material, election master keys, and election constant is used as a "salt" to the commitment function, and is referred to here as "salt with key material," (i.e. Skm).

3.6.2 Commitments

The commitment function uses the AES block cipher with 128-bit key encryption in electronic codebook (ECB) cipher mode, (denoted below as `AES128`). Additionally it uses the SHA-256 hash algorithm denoted below as `SHA256`. The inputs and outputs to this function are base64 encoded so that they may reside in ASCII form in the corresponding XML election data files. However, the cryptographic operations below are preformed at the byte level. Each ballot i has four pieces of information M that are to be committed to; the permutations on the top and bottom sheets (P1, P2) as well as the left and right decryption permutations (D2, D4). Each item within a ballot is committed to separately. The message being committed to (either P1, P2, D2, D4) is denoted as MX, and the commitment of MX denoted as CX.

Algorithm 2: Commitment

Input: MX_i, Skm_i, C
Output: CX_i
1 $Sak_i = \texttt{AES128}_{Skm_i}(C)$
2 $h1_i = \texttt{SHA256}(MX_i || Sak_i)$
3 $h2_i = \texttt{SHA256}(MX_i || \texttt{AES128}_{Sak_i}(h1_i))$
4 $\text{CX}_i = h1_i || h2_i$

3.7 Audit Data Specification

3.7.1 Definition of Terms

Before we can begin explaining the audit process, we need to return to the Punchboard analogy. Recall there are 3 main columns; "prints" table (**P-Table**), the "decryption" table (**D-Table**) and the Results table (**R-Table**). Each ballot i forms one complete row in the Punchboard, but the row number of ballot i's entry differs in the P, D and R tables, as governed by a row permutation. Let ballot i's entry in the Punchboard be located on rows p, q, r of the P, D and R tables respectively.

P-Table

- $P1_i$, permutation on the top page of ballot i
- $P2_i$, permutation on the bottom page of ballot i
- $P3_i$, position of voter's mark on ballot i
- $CP1_i$, commitment to P1 in ballot i
- $CP1_i$, commitment to P2 in ballot i

D-Table

- $D1_i$, pointer to a row p in the P-Table
- $D2_i$, first decryption permutation of ballot i
- $D3_i$, partially decrypted vote on of ballot i
- $D4_i$, second decryption permutation of ballot i
- $D5_i$, pointer to a row r in the R-table
- $CD2_i$, commitment to D2 in ballot i
- $CD4_i$, commitment to D4 in ballot i

R-Table

- R_i, result (i.e. vote) of ballot i

3.7.2 Data Files Used and Produced by the Meetings of the Trustees

During the meetings of the trustees, election and audit data is both used and produced by meetings of the election trustees. Recall there are four meetings of the trustees, and each meeting has an input and output associated with it. The data is contained in XML files that take the name associated with its meeting and function. The meetings files are labeled `Meeting{One, Two, Three, Four}{In, Out}`

Meeting 1 (First Meeting of the Trustees)

During their fist meeting, the trustees instantiate the Punchboard using the election master key. During this time P1, P2, D1, D2, D4, and D5 are generated, but kept secret. The election engine then computes and return the P1, P2, D2, and D4 commitments for all the ballots.

- `MeetingOneIn` specifies how many ballots to generate n, as well as the election constant. Additionally it contains a file pointer to the election specification file which contains the number of contests, as well as the number of respective candidates. This information determines the size of the algebraic group G that will be used to construct permutations and their inverses.
- `MeetingOneOut` contains the commitments CP1, CP2, CD2 and CD4 of all n ballots.

Meeting 2 (Public Disclosure of Pre-Election Audit Data)

During the pre-election audit challenge, the serial numbers of half of the n ballots are selected (randomly) to be "unsealed" and made public for auditing purposes. In meeting 2, the trustees use their passwords to regenerate the Punchboard, and output the secret information of the requested ballots. The ballots are now considered "spoiled" and will not be physically printed or voted on in the election.

- `MeetingTwoIn` contains the list of $n/2$ serial numbers to be audited
- `MeetingTwoOut` returns P1, P2, D1, D2, D3, D4 and D5 for the selected ballots

Meeting 3 (Ballot Decryption and Election Tally)

As the election wraps up and the polling period ends, the ballot sheets scanned at the polling places throughout the election are collected together, decrypted and tallied. Recall when a voter casts their vote, they destroy one sheet of their ballot, and present the other half to be scanned at the polling station. This ballot receipt contains the ballot's serial number i, the position of the mark the voter made (P3) and which sheet they chose to retain (either top or bottom). Once again the trustees enter their passwords, the Punchboard is recreated. The polling place data is entered, and the results are generated. Along with the voting results, meeting 3 also outputs the permutation on the sheet that was retained by the voter (to audit the printing), as well as the intermediate "partially decrypted" vote to allow for auditing of the D-Tables.

- `MeetingThreeIn` essentially contains all the scanned ballot receipts (i.e. encrypted ballots) cast in the election. Specifically, it contains a record of either "top" or "bottom" as having been retained by the voter as well as the mark position ($P3_i$) made by the voter for each ballot i cast in the election.
- `MeetingThreeOut` returns the R table (i.e. the un tabulated election results), D3 and one of either $P1_i$ or $P2_i$ depending if the voter retained the top or bottom sheet respectively.

Meeting 4 (Public Disclosure of Post-Election Audit Data)

Similar to meeting 2, this meeting also responds to an audit challenge. However this time, we cannot reveal the permutations D1 and D5 because this would link serial numbers to their respective entry in the R table (i.e. the vote). Additionally we cannot reveal D2 and D4 for similar reasons. However, the structure of the Punchboard is such that we can reveal either one side of the D-table and not compromise voter privacy. The post-election audit challenge requests either left side of the decryption table D1, D2 or right side D4, D5 for each ballot.

- `MeetingFourIn` contains the side of the decryption table to open (left or right) for each ballot.
- `MeetingFourOut` returns the corresponding tuple, either (D1, D2) or (D4, D5) for each ballot.

Now that we have covered the location and timeline for generating the audit data, we can being discussing the specific checks made against this data during the audit.

3.8 Election Audit Specification

In this section we explain the details behind how Punchscan performs these audits. Broadly speaking, an audit of an E2E voting system should, at minimum, undertake to allow voters to independently verify the following:

1. The ballot data requested for audit by the verifier was indeed the ballot data received from the EA.
2. The ballot data is in accord with its previously posted commitment, meaning that the EA did not alter the ballot given that it was requested for audit.
3. The ballot data is correctly formatted.[1].
4. The number of encrypted ballots input to the decryption function matches the number of decrypted ballots output from the decryption function.
5. The information contained on the ballot receipts retained by voters was indeed the information input put through the decryption function.

[1] This obviously differs from system to system, however in an E2E system, it would likely refer to verifying the correctness of some mathematical relationship (e.g., does 1+1 = 2?

Items 1–4 are verified by the pre and post election audit tools, and item 5 is verified by the online receipt checker. It is required in a Punchscan election that any interested citizen or group be able to personally perform the audit to independently verify its findings. Therefore we must design a system based on notion of availability. The audit must be able to be independently (and repeatably) performed by any interested party on the platform of their choosing. There are four tools that we need to make available for voters to perform the audits. They are:

1. Audit Challenge Generator
2. Pre-Election Audit Tool
3. Post-Election Audit Tool
4. Online Receipt Checker

First is the audit challenge generator which fairly selects ballots to be audited. This could be done by flipping a coin to decide if a given ballot should be revealed and audited. In our implementation we use stock indeces to make these selections in an unbiased, unpredictable but widely verifiable way. Now that we have a means to fairly challenge the EA for ballot data, we also require software tools to perform the cryptographic integrity checks of the pre and post election audits. Finally voters need a means to verify that the information contained on their ballot receipt was actually used in the decyption/tally process.

3.8.1 Audit Challenge Generators

The audit challenge generator for both the pre and post election audits are shown in the `Pre/Post Election Audit Challenge Generator` algorithms below. It uses stock data to generate these random challenges. Stock data is well suited for this task because it is unpredictable *a priori*, yet easily verified *a posteriori*. For each stock in the portfolio, the closing price is concatenated into a long integer. This integer is then used as a seed to a pseudorandom number generator (PRNG). The PRNG does not need to be cryptographically secure because the seed is public knowledge after the stock market closes on the specified date. Our system uses a two-dimensional cellular automata, running rule 30 which is known to have good random properties and adhere to the strict avalanche criteria. The stock portfolio contains 32 stocks—the subset of the *Wired 40* companies [18] which are traded on NASDAQ. The statistical properties of using stocks in this fashion, and their relationship to auditing in Punchscan, has been studied and found secure [7].

Algorithm 3: Pre-Election Audit Challenge Generator

Input: `MeetingOneIn`, Pre-election Stock Portfolio data
Output: `MeetingTwoIn`

```
1 n ← MeetingOneIn.noBallots
2 seed ← ∅
3 foreach stock s_i ∈ Pre-Election Portfolio Data do
4 |   p_i ← ClosingPrice(s_i, timestamp)
5 |   seed ← seed||p_i
6 selection bit stream b ←PRNG (seed)∈ {0,1}^n
7 for i = 1 ··· n do
8 |   if b_i ∈ {0,1} = 0 then
9 |   |   Add serial number i to MeetingTwoIn
```

3.8.2 Pre-Election Audit Tool

Recall at the beginning of this section we explained the requirements for an E2E audit. Relevant to the pre-election audit, we must verify that:

1. The ballot data requested for audit by the verifier was indeed the ballot data received from the EA.

Algorithm 4: Post-Election Audit Challenge Generator

Input: `MeetingThreeIn`, Post-election Stock Portfolio data
Output: `MeetingFourIn`

```
1  n ← Set of all serial numbers in MeetingThreeIn
2  seed ← ∅
3  foreach stock s_i ∈ Pre-Election Portfolio Data do
4  |  p_i ← ClosingPrice(s_i, timestamp)
5  |  seed ← seed||p_i
6  selection bit stream b ←PRNG (seed)∈ {0,1}^|n|
7  for i = 1 ··· |n| do
8  |  if b_i ∈ {0,1} = 0 then
9  |  |  Add serial number n_i and "Left" to MeetingFourIn
10 |  else
11 |  |  Add serial number n_i and "Right" to MeetingFourIn
```

2. The ballot data is in accord with its previously posted commitment, meaning that the EA did not alter the ballot given that it was requested for audit.

3. The ballot data is correctly formatted.

See the following `Pre-Election Audit` algorithm. Lines 2–3 verify item 1 of the above audit criteria. Lines 4–7 verify item 2 of the audit criteria. Finally lines 8–9 verify item 3 of the audit criteria.

Algorithm 5: Pre-Election Audit

Input: `MeetingOneIn, MeetingOneOut, MeetingTwoIn, MeetingTwoOut`
Output: Pre-Election Audit Report

```
1  foreach i ∈ MeetingTwoIn do
2  |  if i ∉ MeetingTwoOut then
3  |  |  Report missing ballot i
4  |  if Commitment(P1_i) ≠ CP1_i then
5  |  |  Report invalid P1_i and/or CP1_i
6  |  if Commitment(P2_i) ≠ CP2_i then
7  |  |  Report invalid P2_i and/or CP2_i
8  |  if P1_i+ P2_i ≠ D2_i+ D4_i then
9  |  |  Report invalid (P1,P2,D2,D4)_i
```

3.8.3 Post-Election Audit Tool

Relevant to the post-election audit, we must verify that:

1. The ballot data requested for audit by the verifier was indeed the ballot data received from the EA.

2. The ballot data is in accord with its previously posted commitment, meaning that the EA did not alter the ballot given that it was requested for audit.

3. The ballot data is correctly formatted.[2].

[2]This obviously differs from system to system, however in an E2E system, it would likely refer to verifying the correctness of some mathematical relationship (e.g., does 1+1 = 2?)

4. The number of encrypted ballots input to the decryption function matches the number of decrypted ballots output from the decryption function.

See the following `Post-Election Audit` algorithm. Lines 11-12 verify item 1 of the above audit criteria. Lines 3–9 verify item 2 of the audit criteria. Lines 13–22 verify item 3 of the audit criteria. Finally, lines 1–2 verify item 4.

Algorithm 6: Post-Election Audit

Input: `MeetingOneIn, MeetingOneOut, MeetingThreeIn, MeetingThreeOut, MeetingFourIn, MeetingFourOut`

Output: Post-Election Audit Report

1 **if** */ballots/ in* `MeetingThreeIn` $\neq$ */ballots/ in* `MeetingThreeOut` **then**
2 Report ballot count error
3 **foreach** $Receipt_i \in$ `MeetingThreeIn` **do**
4 **if** $Receipt_i$ = *top sheet* **then**
5 $PX_i = P1_i$, $CPX_i = CP1_i$
6 **else if** $Receipt_i$ = *bottom sheet* **then**
7 $PX_i = P2_i$, $CPX_i = CP2_i$
8 **if** `Commitment`$(PX_i) \neq CPX_i$ **then**
9 Report invalid PX_i and/or CPX_i
10 **foreach** *side* s_i *(left or right)* $\in$ `MeetingFourIn` *and side* $s_i^* \in$ `MeetingFourOut` **do**
11 **if** $s_i \neq s_i^*$ **then**
12 Report challenge mismatch at i
13 **if** s_i = `Left` **then**
14 $DX_i = D2_i$, $CDX_i = CD2_i$
15 **if** *D3* $\neq$ *P3 + D2* **then**
16 Report invalid decryption
17 **else if** s_i = `Right` **then**
18 $DX_i = D4_i$, $CDX_i = CD4_i$
19 **if** *R* $\neq$ *D3 + D5* **then**
20 Report invalid decryption
21 **if** `Commitment`$(DX_i) \neq CDX_i$ **then**
22 Report DX_i, CDX_i

3.8.4 Online Receipt Checker

This aspect of independent verification is different from the others in that the two items being compared are in different mediums. In the pre and post election audits, we are computing and comparing digital bits. However in this circumstance we are comparing the information contained on a physical object with its electronic manifestation. Recall, in this step the voter checks to see that what they hold in their hand matches what went into the decryption/tally. The algorithm below describes the verification process the voter undertakes. It is important to realize however that the information is displayed in a graphical way, and that although the notation below may seem complicated, the human auditor *is* reasonably expected to be able to notice any inconsistencies. See the image below.

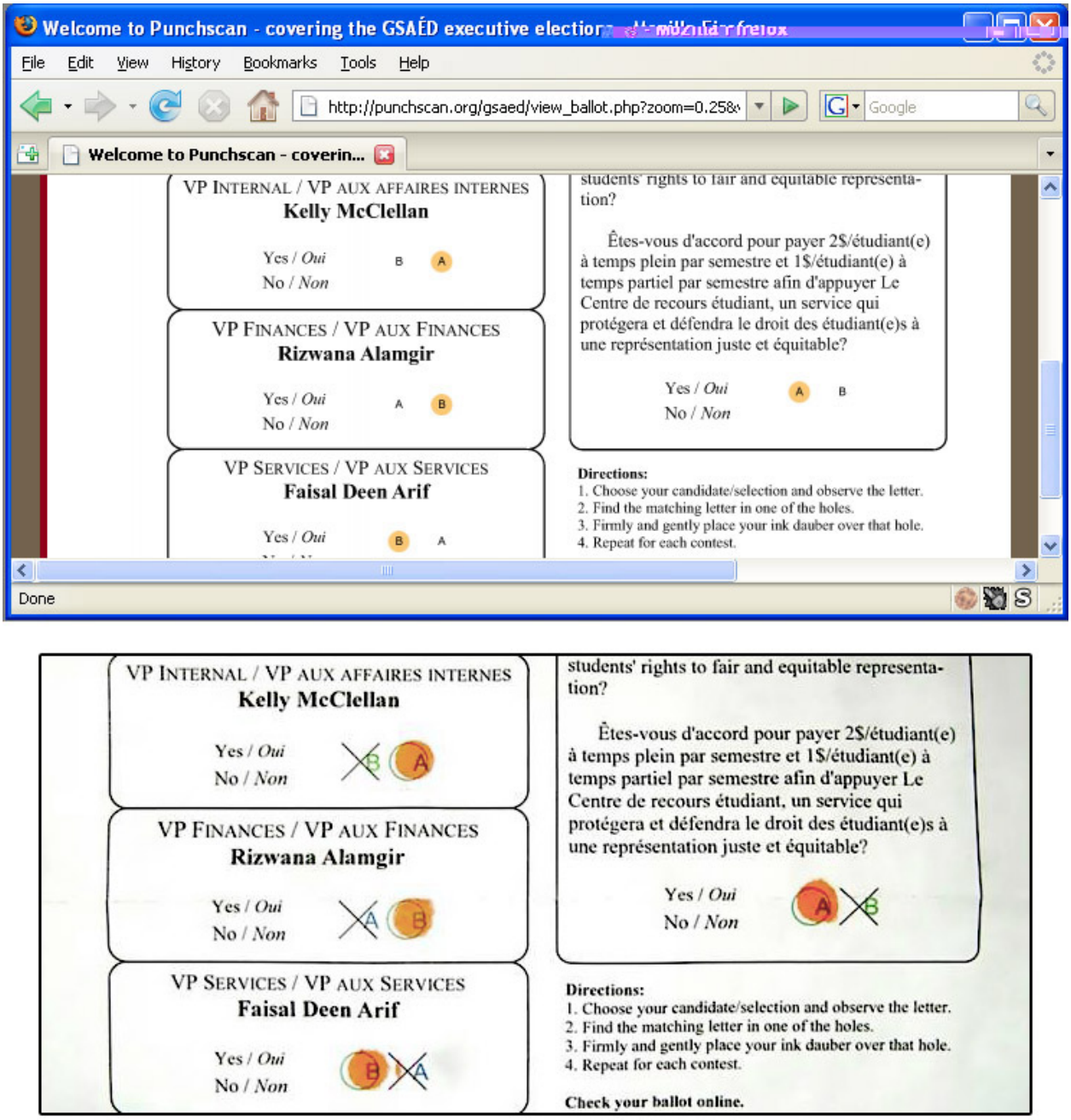


Figure 3.2: (Above) Online Receipt Check showing ballot `#04078` from the 2007 uOttawa GSAED election. (Below) Photograph of the paper original. Visit `www.punchscan.org/gsaed/` and enter serial number `04078` to verify this receipt for yourself.

Algorithm 7: Online Receipt Check

Input: Physical Ballot: Serial number $s^*, P3^*$ and one of $\{P1^*, P1^*\}$ (permutation appearing on either the `top` or `bottom` sheet) respectively

```
1  {s, P3, PX} ← ReceiptCheckingWebsite(s*)
2  if s* ≠ s then
3      Serial number mismatch
4  if P3* ≠ P3 then
5      Mark transcription error
6  if Voter kept top sheet then
7      if PX = P1 then
8          if PX ≠ P1* then
9              Print Error
10     else
11         Wrong sheet recorded
12 if Voter kept bottom sheet then
13     if PX = P2 then
14         if PX ≠ P2* then
15             Print Error
16     else
17         Wrong sheet recorded
```

Chapter 4

Source Code Overview

Source code is available at `http://punchscan.org/vocomp/code/`. There are four zip files in this directory, and they have the following contents:

1. **gsaed-website.tar.bz2** — The election website used for the GSAÉD election (with sensitive data redacted). This includes generic files with information for voters, logs, and the XML files. It also has some cached ballot images and other various data of interest to the website.
2. **database.sql.tar.bz2** — database schema and data for the aforementioned website. This does not include the pollbook database, which contains sensitive data (student ID numbers).
3. **javacode.tar.bz2** – Contains the Audit tools, Ballot Authoring Software, Election Engine, and Polling Place software. Source code written in Java.
4. **selection.tool.tar.bz2** — An optional tool for generating the audit challenges of pre-election ballots and post-election data using the stock market. This includes the selection tools written in Mathematica and instructions on how to use them.
5. **Instructions.tar.bz2** — Detailed instructions on how to run a Punchscan Election.

Chapter 5

Election Procedures

Now that we have defined the functional components of the Punchscan voting system, we can assemble and use them to run an election. In this section we chronicle the steps undertaken in a Punchscan election from start to finish. In addition to the procedures one might expect in a conventional paper ballot or optical scan voting system, an E2E election has additional procedural elements surrounding the verification of the results.

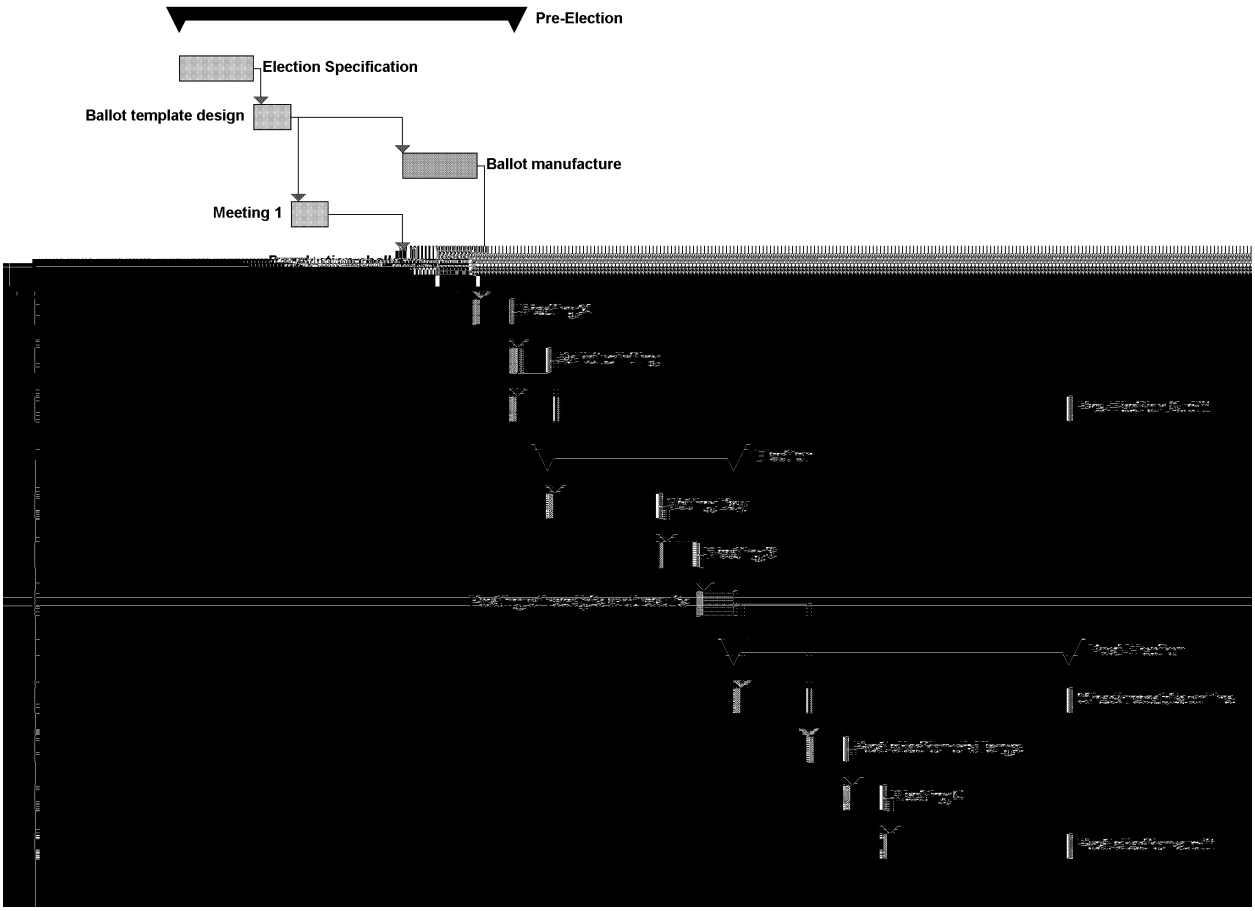


Figure 5.1: **Punchscan election timeline (Gantt Chart).** Task durations are relative. Generally speaking, the meetings of the trustees, audits and audit challenges can be performed on the order of minutes. The duration of the election specification phase, ballot template design and voting days are election specific, and are determined by the needs of the electoral body. Ballot manufacture and printing are on the order of a day, depending on logistics. The lag time of the pre and post-election audit challenges are prescribed to be one business day at minimum. The date up to which voters may perform independent verification via the audits and online receipt checking (and potentially contest the election results) will be mandated by pre-existing election policy/law.

5.1 Trustee Procedure

The process of conducting an election with Punchscan proceeds in four phases, each distinguished by a meeting of the election trustees. At each meeting, the trustees boot a Diskless Workstation with the election software loaded from a removable storage medium (e.g. USB thumb drive), and proceed to validate eachother's instance of the election software against their own. Once the election software has been validated and initialized, each trustee enters a passphrase which authenticates them to the system and seeds a pseudo-random number generator used to generate the Punchboard. Then the data output from the given meeting (as defined in **??**sec:roadmap)) is read from a separate removable device, processed with the verified trusted software and output to a recordable storage medium which is then hand-carried to its destination, often the public Web Server.

5.1.1 Election Definition Phase

In the first of four phases the trustees use the Ballot Template and Ballot Authoring Software to define critical ballot and election parameters, posting the resulting ballot definition file on the Web Server. Once the public has been given a reasonably opportunity inspected the ballot definition file, the trustees will conduct their first meeting.

Officials load the ballot definition file on the Diskless Workstation, which outputs a Punchboard with the specified number of ballots, questions and choice permutations. At this stage, all data is encrypted, but commitments to each data value prevent their alteration by Election Officials. These commitments to the Punchboard are copied from the recordable media to the Web Server.

5.1.2 Pre-Election Phase

Once the commitments to Punchboard are published, and after sufficient time has passed, the pre-election challenge is run using the Audit Challenge tool as defined in 3.8.1. This returns with a list of half the ballot ID numbers listed in the Punchboard. At their second meeting, the trustees again use the Diskless Workstation and their pass phrases to regenerate the Punchboard. The ballots with serial numbers selected for audit by the audit challenge are output to a file and transferred to the Web Server. For each of published (i.e., spoiled) ballots the auditors conduct the pre-election audit as defined in using the open-source/open-spec Pre-Election audit tool.

Also during the second meeting, the Diskless Workstation renders print-ready ballot images for each ballot ID number not chosen for the audit. These ballot images are stored on a separate storage device and transferred to the Printer. Printed ballots are placed in envelopes and transported to each polling place.

5.1.3 Election Phase

On Election Day, each Voter marks their ballot and separates its pages. One page is destroyed, the other scanned. After the Voter verifies the Scanner has correctly detected the marks on their ballot, the ballot is returned to the Voter and an electronic copy is prepared, encrypted and transmitted to the Web Server. A second copy is retained on a removable storage device in case the Web Server or its Internet connection fails. After the polls close, any votes not already transmitted to the Web Server are copied from the removable storage device from each polling place. Voters can visit the election website to verify their ballot is correctly posted on the Punchboard.

Election Officials copy each ballot onto a removable storage device and meet for a third time. The Diskless Workstation fills the Punchboard with data obtained from each encrypted ballot. Each ballot mark is processed through the Punchboard's Decrypt table and stored as a single vote in the Results table. The updated Punchboard and preliminary vote totals are transferred to the Web Server.

5.1.4 Post-Election Phase

After election results are posted online, Auditors perform a Post-Election Audit, choosing either the left or right half of the Punchboard's Decrypt table. Election Officials meet for a final time and use the Diskless Workstation to decrypt the chosen half of the Decrypt table. This step reveals half of the ballot mark

translation process, all intermediate values from this process and the links from each row of the Decrypt table to either the Print or Results table.

From this, Auditors and Observers can verify each calculation in the Decrypt table and that each row of the Decrypt table links to exactly one row of the Print or Results table (and vice versa). Each commitment can be recomputed and compared with earlier editions of the Punchboard to verify the links and data values released to the public have not been altered.

Although the Pre and Post-Election Audits do not reveal all the data and calculations within the Punchboard, they are an effective guard against corruption among Election Officials. Any area of the Punchboard may be opened in response to either audit. In addition, voters may choose either ballot page as their receipt, and any attempt to modify the chosen page is detected when the voter verifies their receipt online. Therefore a corrupt Official must risk detection to alter any aspect of the ballot or the tabulation process. Established formulae for parallel testing and probability theory ensure any significant corruption of the Punchboard is almost certainly detected.

5.2 Voter Procedure

The voter should perform the following steps at the polling place:

1. State her intention to keep the top or bottom page (this can be done by the Poll Worker, so long as the choice is transparent, e.g. picking a ticket from a box that says top or bottom).
2. Retreat to the polling booth.
3. Open the clipboard assembly to view the ballot.
4. For each contest: locate the chosen candidate/option to vote for;
5. Note the symbol beside the candidate/option;
6. Locate the hole containing the same symbol;
7. Mark the hole by firmly daubing it.
8. In view of the poll workers, remove and shred either the top or bottom sheet (as stated in 1).
9. Returned the clipboard with the remaining sheet still locked in to poll workers.
10. Check that the scanned image captures her intent.
11. Retain her receipt as she leaves the polling place.

After leaving the polling place, she is free to distribute copies of her receipt to voting organizations intending to ensure the integrity of the election by checking receipts on voters' behalf. After the election, she is free to do the following:

1. Visit the election website.
2. Type in her ballot serial number.
3. Check that the posted receipt matches her receipt.
4. Audit the pre- and post-election data to assure herself of the end-to-end integrity of the election.

5.3 Poll Worker Procedure

When a voter entered the polling station to vote, the poll workers should perform the following steps:

1. Look up the voter in the voter database.
2. Place the top sheet of a new ballot face down on the clipboard assembly.
3. Place the bottom sheet face down on the clipboard assembly.
4. Close the clipboard assembly so that ballot marks are covered and only the ballot serial numbers are showing.
5. Check that the serial numbers on the top and bottom sheet match.
6. Ask the voter which layer she intends to keep.
7. Lock the ballot in place and give the clipboard to the voter.
8. Provide instructions on how to mark the ballot.

Upon receiving the clipboard, the poll workers should perform the following steps:

1. Unlock the clipboard and remove the remaining ballot sheet.
2. Scan the ballot sheet and display the results of the OCR to the voter.
3. If approved by the voter, cast the ballot.
4. Mark the voter as voted in the database.
5. Place the sheet in the printer for the overlays, digital signature, and paper backup.
6. Cut the paper backup off the bottom corner of the sheet.
7. Give the sheet to the voter as a receipt.
8. Place the paper backup in the ballot box.

If the voter returns with a ballot that has been removed from the lock or a ballot on which they have made a mistake, they can be reissued a ballot following the same procedure. Typically the number of tries is limited, and in the case of a busy polling place, a retry can be marked by locking a washer in with the ballot for each successive try.

5.4 Step-by-step Instructions

This section outlines step-by-step instructions for running a Punchscan election using the available software.

5.4.1 Preparation

1. In the `C:\` directory, create a folder called `MockPunchScanElection`.
2. Go to the following webpage:
   `http://punchscan.org/~stefan/new/`
3. Download the "ballot" and the "background" from the webpage to:
   `C:\MockPunchScanElection`

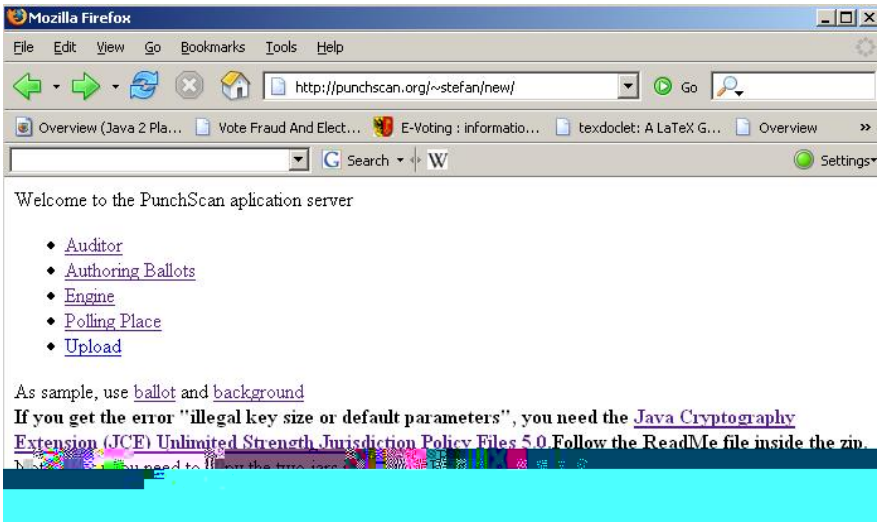


Figure 5.2: The webpage showing the ballot and background for download.

5.4.2 Producing the Layout Of the Ballot

1. On the web page, click on the `Authoring Ballots` link. Instruct the browser to open the link with Java(TM) Web Start Launcher (see Figure 5.3). If you do not have this option, then download and install Java 5.0 (which includes the needed Java JRE 1.5). If you have another version of Java already installed, it will prompt you to install the Java 1.5 JRE. This version is required. After JRE 1.5 has been installed, download and install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 5.0 from:
   `http://java.sun.com/javase/downloads/index_jdk5.jsp`
   Follow the instructions in the `ReadMe` file.

2. Through the menus, choose `File`→`Load Fully Marked Ballot`. Go to the `MockPunchScanElection` directory and choose `ballot.jpg`. A PunchScan Ballot will appear in the main window (if no image initially appears, resize the window slightly to force a repaint).

3. Choose the requested colors by checking the radio button and then clicking on the picture in the main window (e.g., click on the alignment radio button and click where the alignment mark is on the picture in the main window, etc.). See Figure 5.4. Click `Done` when finished.

4. Enter 1 when asked how many columns (see Figure 5.5).

5. When the "Done" message appears, two files have been created in `C:\MockPunchScanElection`. `ElectionSpec.xml` represents the types of questions the ballot has and `geometry.xml` contains the layout of the ballot.

6. Go to `Tools`→`Form Maker`. For the background, choose `background.pdf` from the `MockPunchScanElection` directory and as the destination folder, choose `MockPunchScanElection` directory. A PDF called `javaCreatedForm.pdf` is created in the directory.

7. Close the Authoring Ballots program.

5.4.3 Running Meeting One

1. Create two folder in `MockPunchScanElection`: one called `public` and the other `private`.

2. Go to the web page and click on `Engine`. The screen in Figure 5.6 should appear.

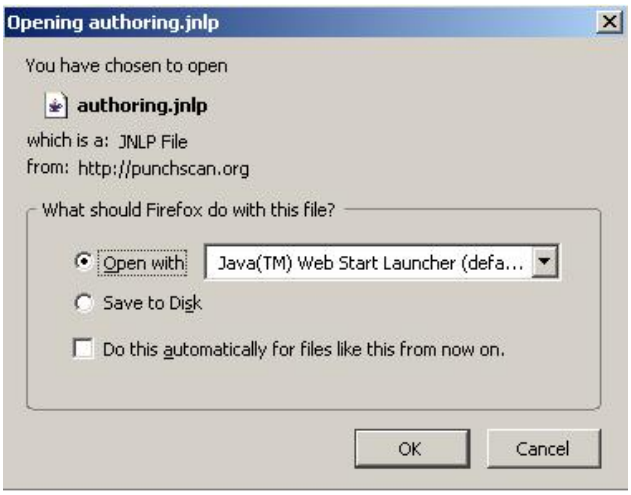


Figure 5.3: The JNLP dialogue.

3. Click on `Initialize Election (Meeting One)`. As the private folder, choose `C:\MockPunchScanElection\private`.
   As the public folder, choose:
   `C:\MockPunchScanElection\public`

4. If you get the error "illegal key size or default parameters," make sure you have completed Step 1 of Section 5.4.2.

5. If no input file for Meeting One is found, the software will ask for the information needed to create it. As a public constant, enter "PunchScanElection," enter 100 for the number of ballots, and 3 for the number of D-tables (see Figure 5.7). To choose the election specification, click `Browse` and select:
   `C:\MockPunchScanElection\ElectionSpec.xml`

6. Click save and exit. A file called `MeetingOneIn.xml` is created in:
   `C:\MockPunchScanElection\public`

7. Enter "PunchScan" as the username and "1234" as the password, and click done.

8. The message "Running Meeting 1...Done" will appear after a few seconds (see Figure 5.8). A file named `MeetingOneOut.xml` will be created in the `public` directory.

5.4.4 Running Meeting Two

1. In the `Engine` window, click on `Pre-election Audit (Meeting Two)`. For the private folder, choose:
   `C:\MockPunchScanElection\private`

2. For the public folder, choose:
   `C:\MockPunchScanElection\public`

3. Enter "PunchScan" as the username and "1234" as the password and click done.

4. If no input file for Meeting Two is found, the software will ask if you want to generate random input (the challenged ballots are chosen by a random number generator). See Figure 5.9. Select yes.

5. In a couple of seconds, the message "Running Meeting 2...Done" will appear. The files `MeetingTwoOut.xml` and `SerialMap.xml` will appear in:
   `C:\MockPunchScanElection\public`

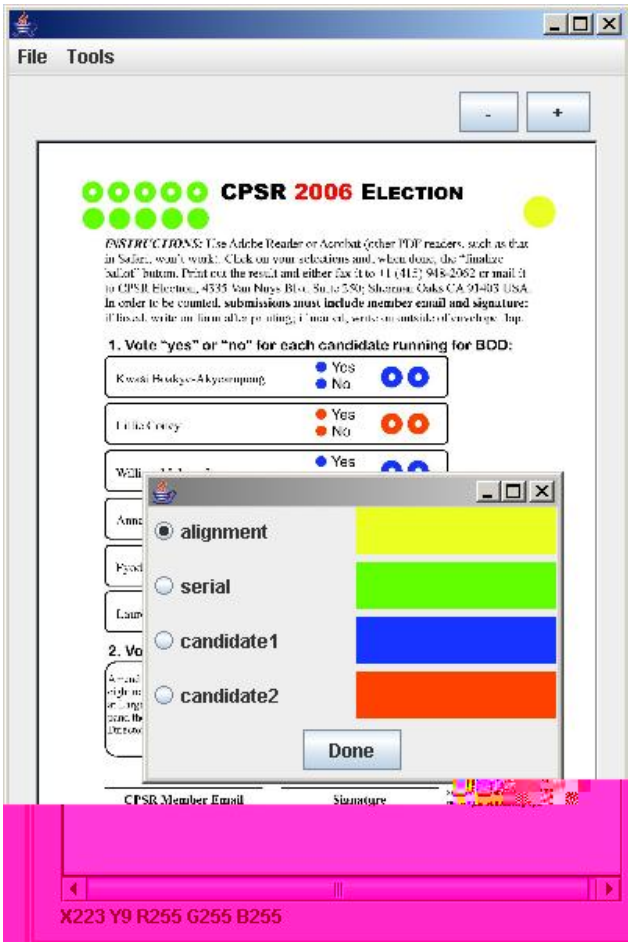


Figure 5.4: The color selection dialogue.

6. A file named `Prints.xml` will appear in:
   `C:\MockPunchScanElection\private`

7. Close the engine.

5.4.5 Running the pre-election audit

1. Go to the web page and click on Auditor.

2. As MeetingOneIn, select:
   `C:\MockPunchScanElection\public\MeetingOneIn.xml`

3. Do likewise for meeting 1 out and meeting 2 in and out (see Figure 5.10). Leave meetings 3 and 4 empty.

4. Click PreElection Audit.

5. When complete, the message "The auditor was successful" will appear.

6. Close the auditor.

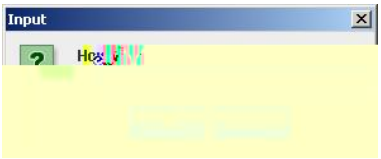


Figure 5.5: The column number dialogue.

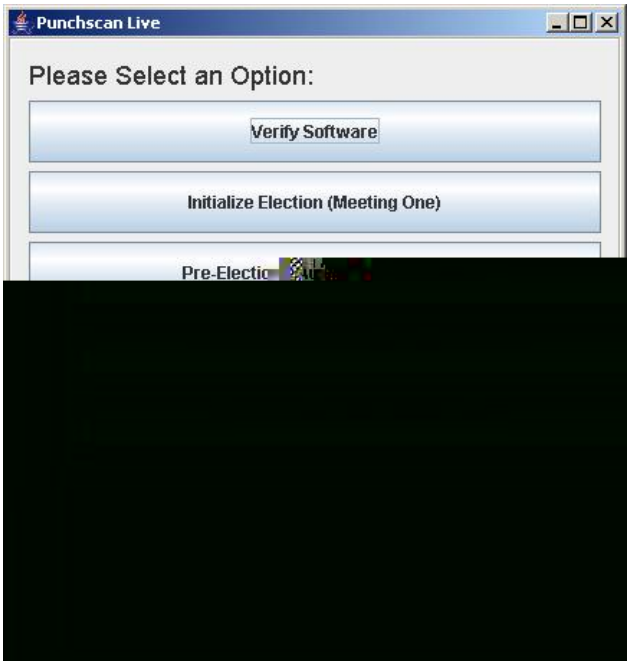


Figure 5.6: The Punchscan Engine menu.

5.4.6 Creating Ballots

1. Go to the web page and click on `Authoring Ballots`.
2. Select from the menu `File`→`Load PDF Form` and select:
   `C:\MockPunchScanElection\javaCreatedForm.pdf`
3. A PunchScan ballot will appear.
4. For the election spec, choose: `C:\MockPunchScanElection\ElectionSpec.xml`
5. For the prints file, choose: `C:\MockPunchScanElection\private\Prints.xml`
6. Select from the menu `Tools`→`Ballot Maker`.
7. Enter "0-10" when asked for the serial numbers (see Figure 5.11).
8. For the output folder, choose:
   `C:\MockPunchScanElection\private`
9. 11 PDF ballots will appear in:
   `C:\MockPunchScanElection\private`

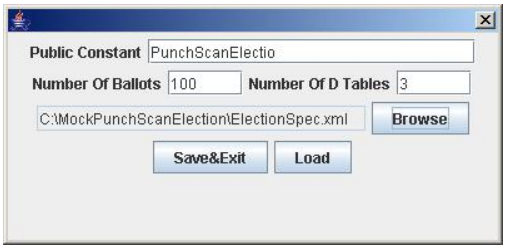


Figure 5.7: The Meeting 1 Configuration Dialogue.

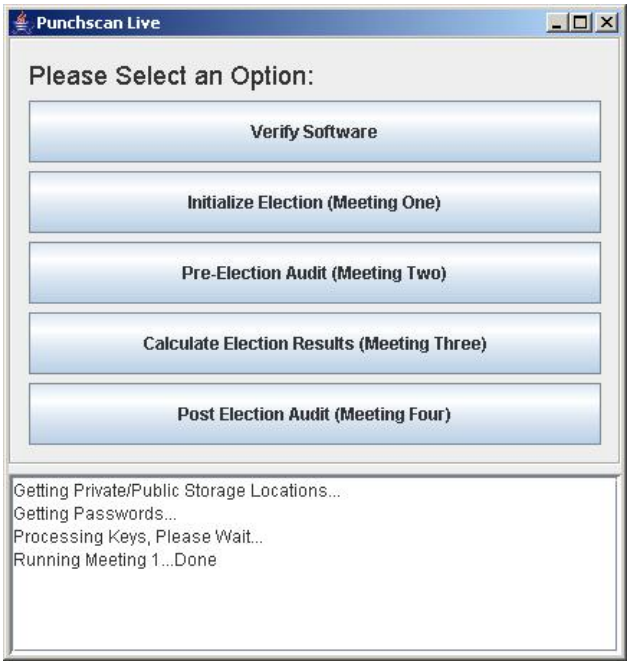


Figure 5.8: After successful completion of Meeting 1.

5.4.7 Voting

1. We now need scanned images of voted ballots. The ballots can be printed, and then scanned after voting for a real election. However for testing the software, it is easier to use a virtual printer capable of printing documents to images. We recommend PdfCreator [1].

2. Create a folder called `scannes` in:
   `C:\MockPunchScanElection\`

3. Open:
   `C:\MockPunchScanElection\private\2.pdf`

4. Vote on the ballot by clicking over Yes or No for each contest. You may leave some contests unvoted. See Figure 5.12

5. When done selecting choices, click Finish Selection. Choose left or right (see Figure 5.13. Print the ballot using PDF creator as a bmp (see Figure 5.14), and save it in:
   C:\MockPunchScanElection\scannes

[1] `http://sourceforge.net/projects/pdfcreator/`

Figure 5.9: The option to generate random input for Meeting 2.

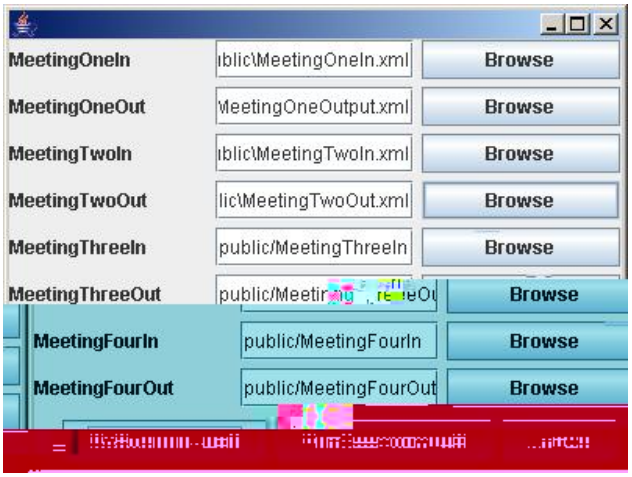


Figure 5.10: The Auditor interface.

6. Do likewise with three other ballots.

5.4.8 Scanning Ballots

1. Go to the web page and click on Polling Place.
2. Create the two folders `ballots` and `ballotsBackup` in:
   `C:\MockPunchScanElection`
3. Leave the password fields blank.
4. For "The images from the scanner will appear in,"
   Browse→`C:\MockPunchScanElection\scannes`
5. Check "Scan existing images."
6. For "The scanned ballots will appear in the folowing folder,"
   (a) Browse→`C:\MockPunchScanElection\ballots`

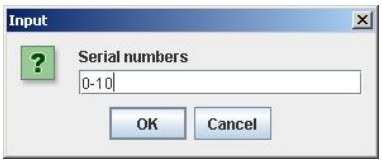


Figure 5.11: The serial numbers to be produced on the ballots.

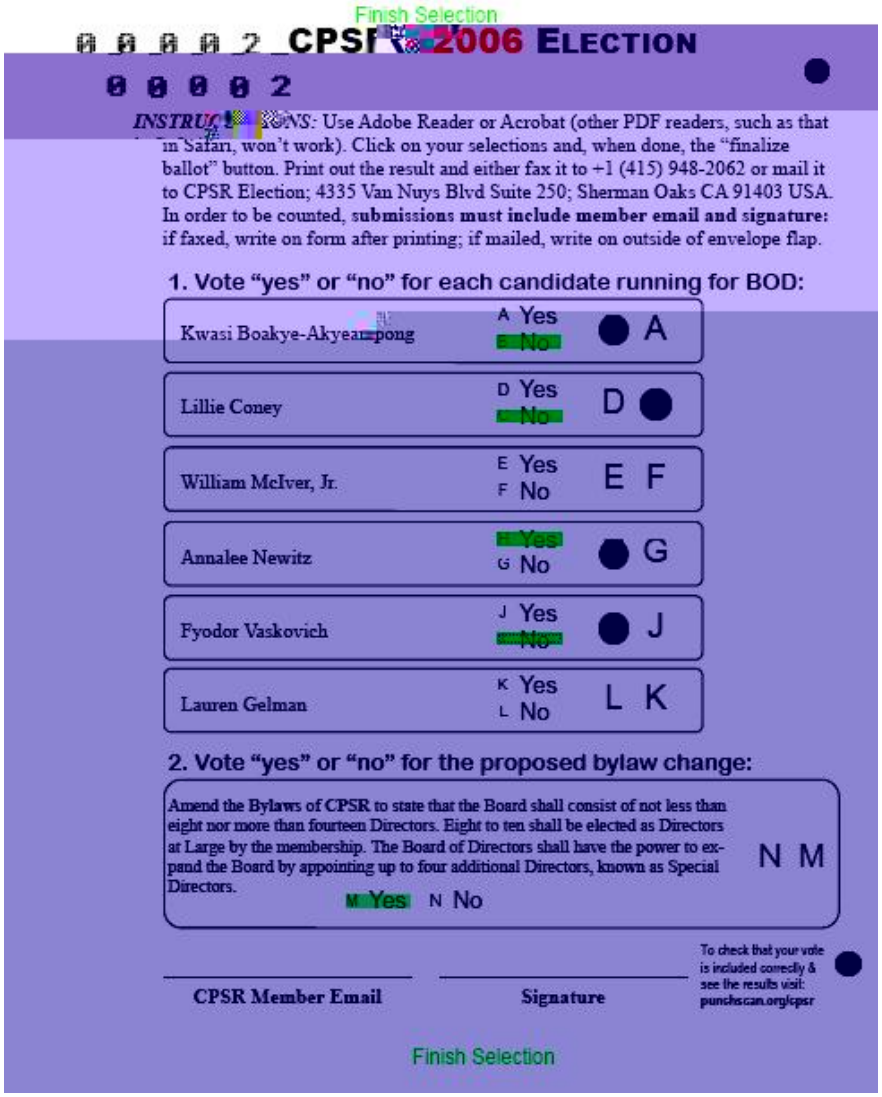


Figure 5.12: A Punchscan virtual ballot.

Figure 5.13: Choosing which layer of the ballot to keep.

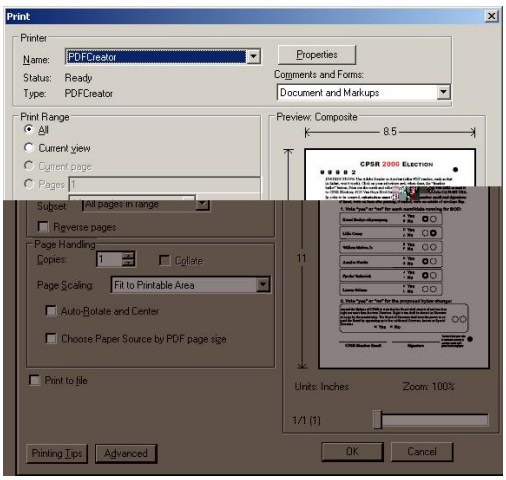


Figure 5.14: The Print dialogue showing PDFCreator.

(b) Browse→`C:\MockPunchScanElection\ballotsBackup`

7. For "Geometry,"
   Browse→`C:\MockPunchScanElection\geometry.xml`
8. For "Election Specification,"
   Browse*rightarrow*`C:\MockPunchScanElection\ElectionSpec.xml`
9. See Figure 5.15. Click OK.
10. The four ballots that you voted appear one after the other on the voter screen (see Figure 5.16. Verify if the marks and the serial number have been detected correctly and cast them.
11. Close the Polling place program.

5.4.9 Running Meeting Three

1. Go to the web page and click on `Engine`.
2. Click on `Calculate Election Results (Meeting Three)`. As the private folder, choose:
   C:\MockPunchScanElection\private
3. As the public folder, choose:
   `C:\MockPunchScanElection\public`

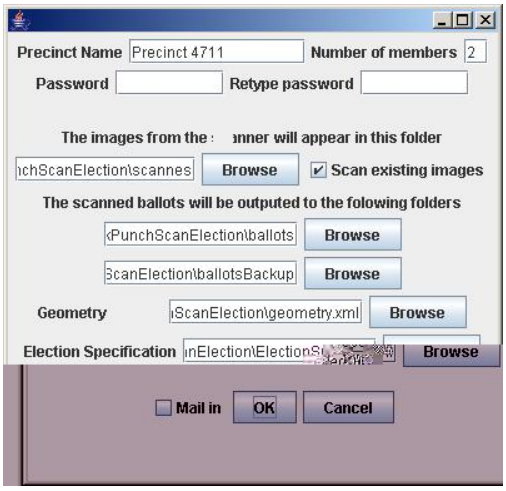


Figure 5.15: Polling Place configuration dialogue.

4. Enter "PunchScan" as the username, "1234" as the password, and click done.

5. If no input file for Meeting Three is found, select "Read Ballots from Folder" (see Figure 5.17) and select:
   `C:\MockPunchScanElection\ballots`

6. After a few seconds, the message "Running Meeting 3...Done" will appear. A file named `MeetingThreeOut.xml` will appear in:
   `C:\MockPunchScanElection\public`

5.4.10 Running Meeting Four

1. In the Engine window, click on `Post Election Audit (Meeting Four)`. For the private folder, choose:
   `C:\MockPunchScanElection\private`

2. For the public folder, choose: `C:\MockPunchScanElection\public`

3. Enter "PunchScan" as the username, "1234" as the password, and click done.

4. If no input file for Meeting Four is found, the software will ask you if you want to generate random input (see Figure 5.18). Select yes. In this case, each row in the D table will have either its left or right side opened, as selected by a random number generator.

5. You will be asked if you want to open all the unused ballots. Select yes. This is used as an extra check.

6. Momentarily, the message "Running Meeting 4...Done" will appear. A file named `MeetingFourOut.xml` will appear in:
   `C:\MockPunchScanElection\public`

7. Close the engine.

5.4.11 Running the Post Election Audit

1. Go to the web page and choose `Auditor`.

2. For MeetingOneIn, select:
   `C:\MockPunchScanElection\public\MeetingOneIn.xml`

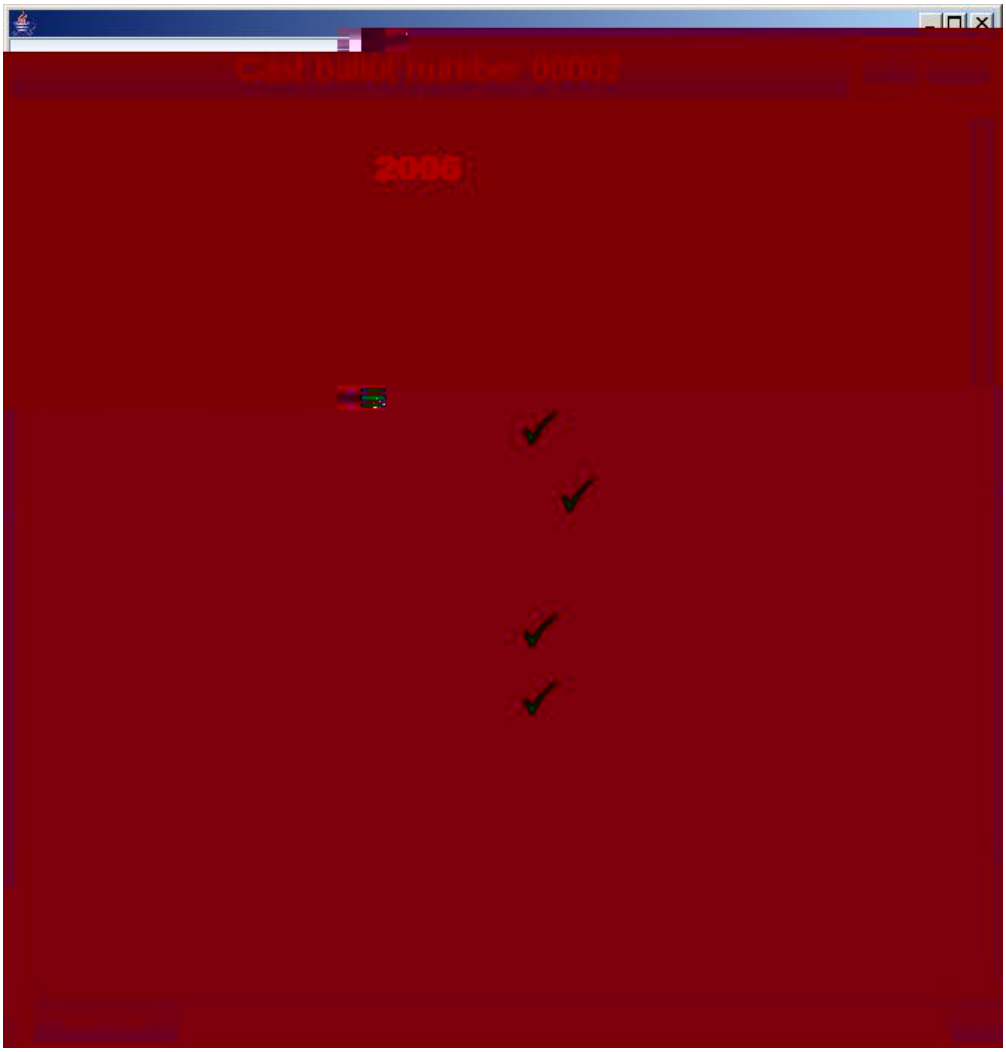


Figure 5.16: OCR recognition of a scanned ballot.

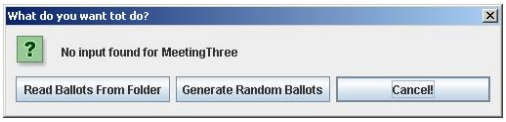


Figure 5.17: Meeting 3 input generator.

Figure 5.18: Meeting 4 random input generator.

3. For MeetingOneOut, select:
   `C:\MockPunchScanElection\public\MeetingOneOut.xml`
4. Delete everything in MeetingTwoIn.
5. For MeetingTwoOut, select:
   `C:\MockPunchScanElection\public\MeetingFourAllUnvotedRevealed.xml`
6. For MeetingThreeIn, select:
   `C:\MockPunchScanElection\public\MeetingThreeIn.xml`
7. For MeetingThreeOut, select:
   `C:\MockPunchScanElection\public\MeetingThreeOut.xml`
8. For MeetingFourIn, select:
   `C:\MockPunchScanElection\public\MeetingFourIn.xml`
9. For MeetingFourOut, select:
   `C:\MockPunchScanElection\public\MeetingFourOut.xml`
10. Click `PreElection Audit`, which checks that the unused ballots have been correctly opened.
11. Click `PostElection Audit`, which checks if the voted ballots have been correctly tallied.
12. Momentarily, the message “The auditor was successful” will appear.
13. Close the auditor.

Chapter 6

Campus Election Results

The University of Ottawa's *Graduate Students' Association / Association des étudiant(e)s diplômé(e)s* (GSAÉD) voted unanimously to adopt the Punchscan voting system for their 2007 executive election held over the three day period of March 6–8. Among the reasons they cited for their decision was the desire to speed up the tally process, increase the integrity of election results, provide a means to identify double-voting, and, at an academic level, play a leading role in voting systems research.[1]

The GSAÉD's chief returning officer (CRO) in conjunction with the Punchscan team proceeded to conduct the first binding election with independently verifiable results.

6.1 Election Requirements

In consultation with GSAÉD, several requirements for this election were arrived upon. GSAÉD required there to be five polling stations located on the University of Ottawa campus, which were to be operated during normal business hours across a three day period. Two poll workers were to be present at all times. Being a fully bilingual university, the ballot was required to be worded in both English and French, and it was decided that this would be accomplished through a single bilingual ballot, as opposed to separate ballots for each language. The offices, candidate names, as well as French translations were provided by GSAÉD.

In addition to these basic requirements, it was agreed that several other additions over the basic Punchscan system would be implemented to provide greater security and robustness. They are as follows:

6.1.1 Ballot Receipt Digital Signatures

An E2E ballot receipt allows a voter to verify their vote was counted correctly, but also serve as proof when it was not. At it's core it protects the voter from mistakes or malicious activity on the part of the election trustees. However given that Punchscan ballots are produced using low cost drilling and printing methods, fabricating a counterfeit receipt for the purposes of invalidating the election results is entirely feasible. Therefore we need a means to protect the election trustees (and in turn the election results) from malicious voters. This threat was addressed by the introduction of a barcode-based digital signature printed onto the ballot receipt at the time that it was cast. At the polling place, the ballot receipt is scanned and the Polling Place software detects and encodes the serial and mark positions into an XML file. The ballot receipt is then placed in a printer and green 'O' shaped overlays are printed over top of letters marked by ink daubs to confirm and 'lock-in' the location of the daub mark. Additionally 'X' shaped overlays are printed over unmarked positions. The marked positions are then digitally signed and printed on to the bottom left corner of the ballot receipt in a scanable barcode format, allowing the election trustees to establish the authenticity of a ballot receipt in the event of a future dispute.

[1] This chapter reprinted in large part from [8].

Figure 6.1: A pollworker explains to a voter how to mark a Punchscan ballot during the GSAÉD election in Ottawa.

6.1.2 Paper-based Backup

Punchscan records ballot marks by optically scanning and electronically storing the marks. Until this point however, GSAÉD had only ever employed traditional (i.e. strictly paper) ballot and had procedures for their handling, counting, and eventual disposal. Because Punchscan is an optical scan system, and because the only surviving physical portion of the ballot (i.e. the receipt) is retained by the voter, the concern was raised by GSAÉD over keeping a strictly electronic record of the election. For example, their election procedure stipulates a date (after the election results have been ratified) by which the election ballots are to be shredded. Other concerns expressed by the council included power outages at the polling stations, accidental deletion, hardware/hard-disc failure, and other electronic attacks. The Punchscan team consulted with them at length about a means to provide a paper-based ballot backup. It was agreed that a small space in the lower right-hand corner of the ballot was to be reserved for printing an encoded copy of the marked positions, which was performed at the same time as the overlays/digital signature. Using a pair of scissors, the poll worker cut off and retained this corner before returning the receipt to the voter. These corners were collected and retained by GSAÉD functioning in the stead of actual ballots.

6.1.3 Electronic Pollbook

In past GSAÉD elections, voters were authenticated at the polling place using local paper copies of the voters list. Voters' university-issued graduate student cards were used as the primary credential to vote. Although this scheme allows the eventual identification of students who cast more than one ballot (so-called "double voting") at different polling stations, it does not allow a means by which to invalidate the double votes from the tally. GSAÉD had experienced problems with double voting in past elections and had employed solutions such as the use of walkie-talkie radios to communicate the student numbers of voters in a real-time fashion. However the local copies of voter lists were still being updated manually which is inefficient given a list of 4000 eligible voters and 5 polling stations.

The Punchscan team offered and implemented an alternative solution: a centralized electronic pollbook. A MySQL database of eligible student numbers was created and maintained on the `punchscan.org` server. The poll worker interface was through any standard web browser and internet connection. Using the campus wireless network, the poll workers were able to log into a password protected, SSL secured web form and perform voter list queries and updates. Querying a voter's student number would return one of three possibilities: a message indicating that the student number was not found in the database, an option to mark that individual as having voted, or an alert that that student had voted already. The pollbook webpage was accessed initially by an SSL connection

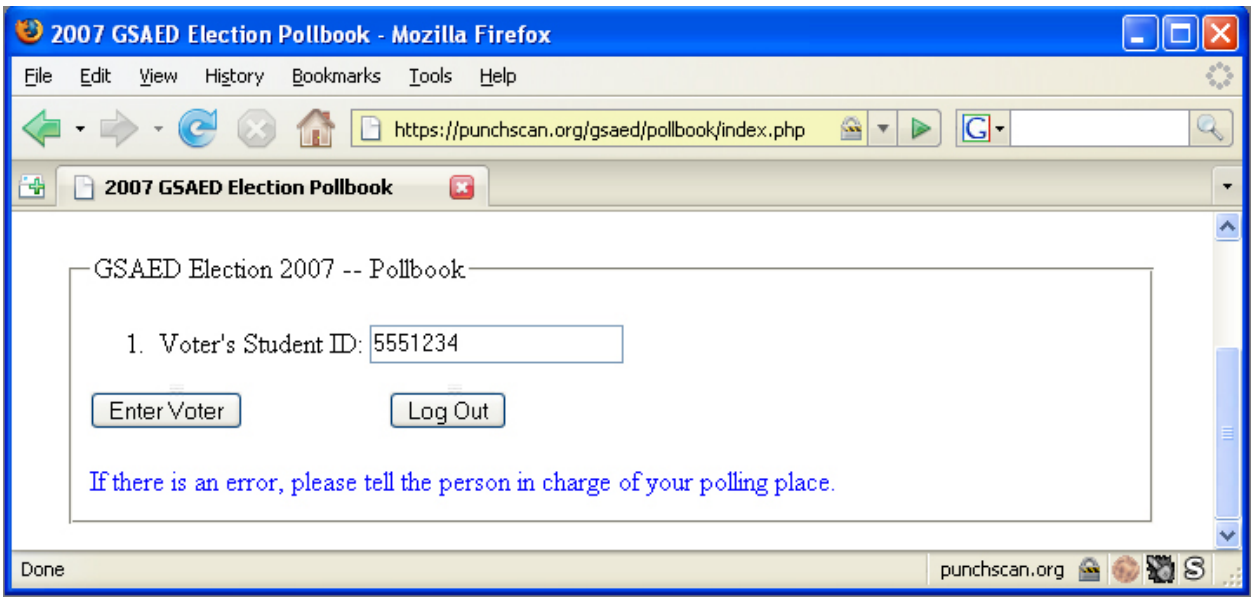


Figure 6.2: Online Electronic Pollbook used in GSAÉD Election

6.1.4 Ballot Clipboard and Lock

Although it was agreed to be unlikely to occur in this election environment, the Punchscan team decided to test a countermeasure to the vote buying/intimidation technique known as "chain voting." If a chain voting perpetrator is able to remove one uncast ballot from a polling station without detection, they could, using the cooperation of subverted voters, "require that the subverted voter take the ballot to a polling place, exchange the pre-marked ballot for the blank ballot issued to that voter at the polling place, and return the blank ballot to the perpetrator to enable the next cycle" [10].

To increase the difficulty of someone being able to remove a ballot from the polling station without detection, the Punchscan team developed special clipboards that employ a *Medeco* plunger lock affixed to the clipboard. When the locking mechanism (located in the upper-right corner) was depressed, the lock's bolt would extend down into a hole in the clipboard. Each ballot also had a hole drilled in the top right corner. Before the voter is presented with an unmarked ballot it is placed on the clipboard. When the lock cylinder is depressed, the plunger would seamlessly pass through the ballot and clipboard locking it into place. When the voter returns to cast their vote, the poll worker checks to ensure that the ballot is still locked to the clipboard and that the paper corner is not torn. Additionally the clipboards were fitted with a cover to obscure from view the unique information printed on the ballot.

6.2 Preparing for the Election

There were approximately 4000 registered graduate students at the University of Ottawa, all of which were eligible to vote. Voter turnout in past years was quite low (about 5%), and even with optimistic estimates it was not realistically expected to exceed 10% turnout for this election. Nevertheless we decided to err on the side of caution, resolving to print 3000 ballots.

6.2.1 Ballot Manufacture

Ballot manufacture refers to the process of creating holes in the appropriate locations of the physical ballot stock. However drilling holes happens to be a reasonably simple, inexpensive and available process in this application. When ballot authoring software recognizes the ballot template markers (discs and rings), it produces a special XML formated "drill file" which is used by a machinist to accurately drill holes in reams of 500 sheets of paper at a time using a computer-positioned vertical mill. As per their election procedure,

Figure 6.3: Ballot lock and clipboard used in GSAÉD election

the GSAÉD council met one week before the election to approve the ballot wording and layout. That evening the Punchscan team produced the drill file and took it to the machinist who drilled 3000 blank ballots. The ballots were then shipped and arrived in Ottawa the next morning.

6.2.2 The First Meeting of the Trustees

The Punchscan team implemented the threshold based secret sharing scheme previously described in this document, however given the stakes in this election were relatively low, GSAÉD exploit the option of distributing the key, and did not produce a trustee in addition to the CRO, who acted in this election as sole trustee.

At the first meeting the CRO supplied her passphrase and the open source election engine software generated all the election data from it. This includes the association of the ballot serial numbers with a pseudo-randomly generated permutation of letters. Additionally, the information needed to reconstitute, or "decrypt" the vote after one half of the ballot is destroyed is also generated. This "decryption table" will be partially revealed during the post-election audit. However to increase the probability of catching tampered ballots during the audit, more (independent) decryption tables can be created, each with a unique keystream. For this election we used 10 such tables. After this secret information is created, each piece is run through a bit commitment function. These commitments are publically posted. Later as some of this secret information is revealed during the audits, it can be run through the same commitment function (by anyone) to ensure it matches the original commitments, establishing that the results have not been modified. These commitments, and all the election data referred to throughout this case study, are available from the Punchscan webpage [2].

6.2.3 Pre-Election Audit

After all ballots are (digitally) generated and committed to, half of the ballots are chosen at random to be audited. Because we ultimately wish to print 3000 ballots, during the election specification we (digitally) generated 6000 ballots.

[2] http://punchscan.org/gsaed/

The pre-election audit took place in two stages. First the CRO entered her passphrase to open and publish the information about the selected ballots. After this data had been published, the second stage was to actually audit this information against the published commitments from the first meeting. The audit report generated by the program determined that all the revealed ballots matched their commitments. It is important to note however that the audits form the core of the "independent verification" and is meant to be as available and performable by anyone interested.

6.2.4 Printing Ballots

After the completion of the pre-election audit, the CRO met with the Punchscan team to print the remaining 3000 ballots on paper. Under the supervision of the CRO, the ballots were printed in parallel on six inkjet printers. Three printers were loaned by Hewitt Packard to the University for the election (two HP-K5400's and one HP-K550's), and three more were provided by the Punchscan team (HP-K550's). Although strictly speaking the printing is not dependent on a particular model of printer, a few considerations need to be taken. Firstly, the ballots contained colored scanner-alignment marks. Secondly the ballots must be accurately aligned so that the serial number and letters on the bottom page show through the holes. This is usually resolved through trial and error, and using the ballot authoring software to introduce a compensation in the ballot PDF. Thirdly it was discovered for certain models that printing would always halt when the print head reach a drill hole. Finally paper feeding was an issue because the drilling process creates a small cusp in the paper around the drill hole causing the sheets to "stick" together as they were fed into the printer. This often results in numerous pages being scrapped. However most of these issues had been previously addressed and the printing process for the 3000 election ballots took approximately one hour. Upon completion, the ballots were placed into boxes, sealed, and signed along the seal by the CRO.

6.2.5 Poll Worker Training

On the night prior to the election, the Punchscan team hosted a two-hour poll worker training program. For the first hour, an introduction to Punchscan was given explaining the theory behind the system, outlining the polling place procedures (which was also provided in written form), and answering questions. The second hour provided hands-on experience with the polling place equipment. The poll workers were provided with mock ballots to vote on and they practiced the procedure of scanning and casting the ballots.

6.3 Conducting the Election

6.3.1 Contests

The GSAÉD election consisted of six contests. Five were positions for office and one was a referendum. Of the five office positions, only one was contested. However the uncontested officials still needed to be confirmed by a majority of voters according to the GSAÉD regulations. Thus these four contests consisted of a 'yes' or 'no' option. The contested office position had two candidates, and the referendum also consisted of a 'yes' or 'no' option. In essence, the election consisted of five contests, all of which had two candidates/options.

6.3.2 Polling Station

For this election, the polling station equipment was provided by the Punchscan team to GSAÉD at no charge. There were five polling stations on each of the three days of the election. Four of the stations were distributed across the main campus, and one was hosted at a satellite campus. Each polling place consisted of a computer, printer, scanner, paper shredder, polling booth, a couple ink daubers, and a ballot box.

6.3.3 Voting and Casting

The ballot-issuing, marking and casting procedures were conducted in accordance with those presented in the previous chapter on election procedures.

6.4 After the Election

6.4.1 Election Results

After the polling stations were closed, the Punchscan team, the CRO, and a scrutineer assembled to tally the results. The encrypted ballot receipts were uploaded to the Punchscan server. The CRO then entered her passphrase to decrypt and tally the results. With 154 ballots cast, the results are summarized in Table 6.4.1.

	President M. Doumit	VP Internal K. McClellan	VP Finances R. Alamgir
Yes / Candidate A	118	120	117
No / Candidate B	28	21	31
	VP Services F. Deen-Arif	VP Communication A) P. Marchand B) T. Plante	Referendum
Yes / Candidate A	121	81	98
No / Candidate B	26	54	43

Table 6.1: GSAÉD 2007 election results

6.4.2 Online Receipt Check

At the bottom of each ballot was instructions (in English and French) directing the voter to the online receipt check website. There voters could enter the serial number of their ballot to verify what they were holding in their hands (i.e. the ballot receipt) is in fact what was used in the tally.

6.4.3 Post Election Audit

After the completion of the election, the tallying procedure was audited. The 10 decryption tables were each partially revealed, again depending on selections made by stock market data in similar form to the pre-election audit. The CRO entered her passphrase in order to recover the partial election data to be published online. This data was then audited with the Post-election audit software tool provided by Punchscan team. The post-election audit verified the election data and encountered no errors or irregularities.

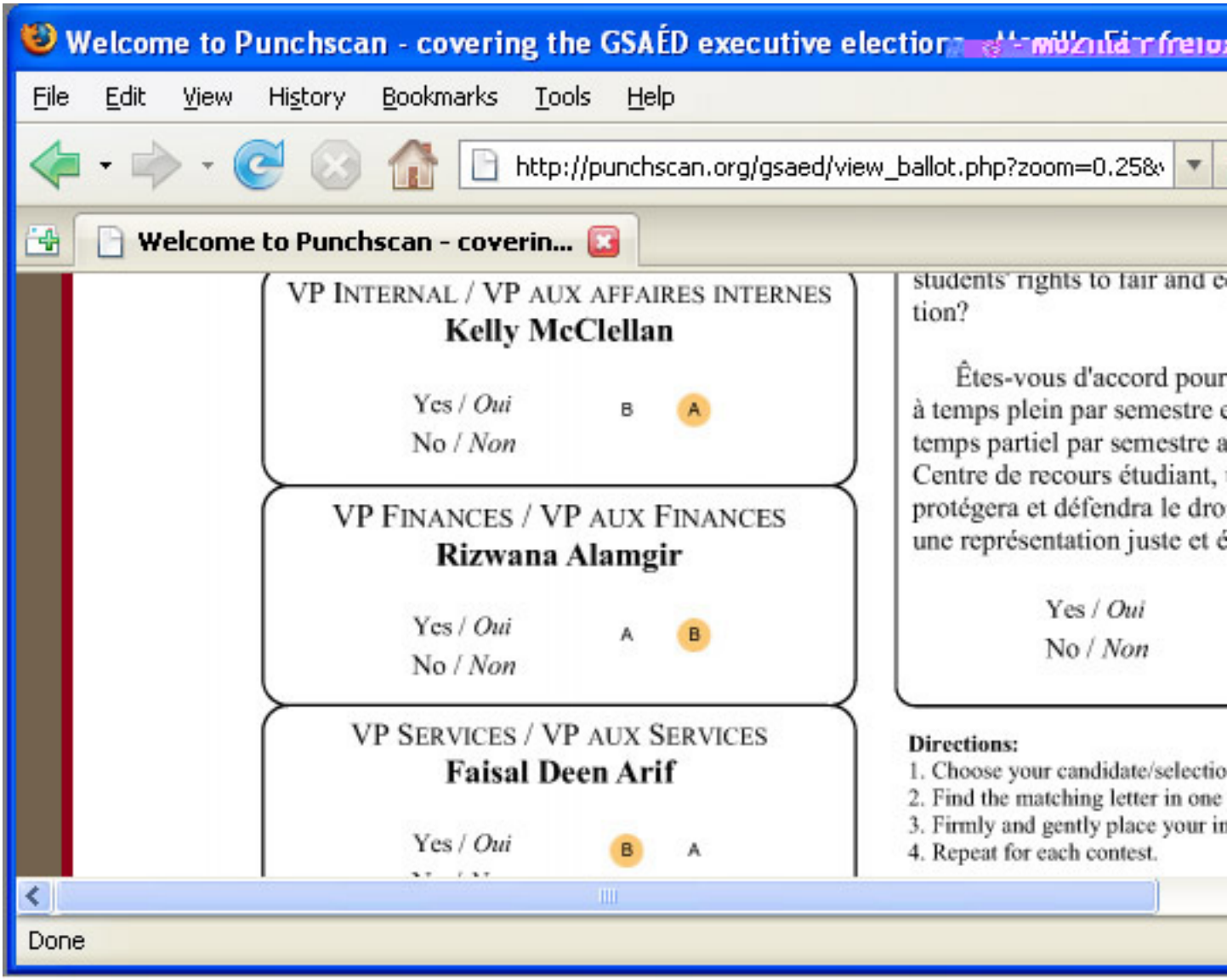
6.5 Incidents and Assistance Requests

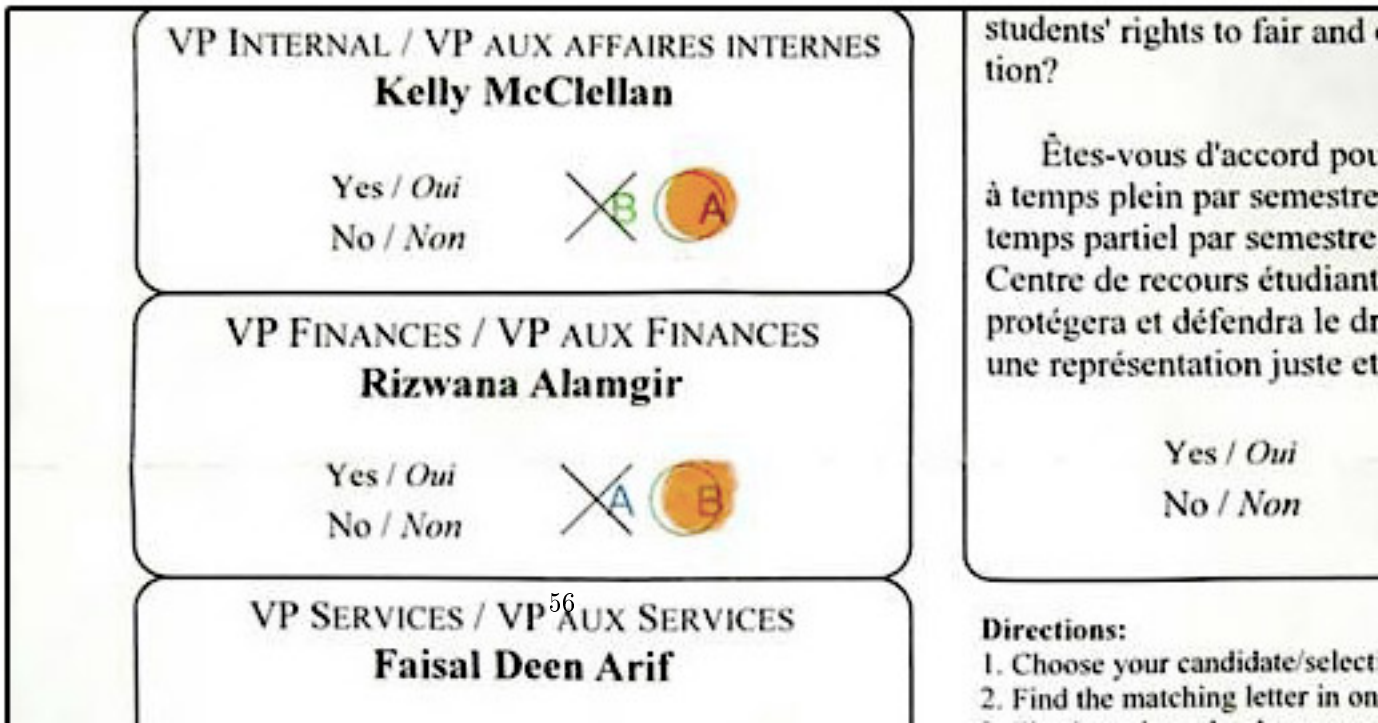
In short, this election fared exceptionally well. Considering the novelty of the software and voting process, we were pleased to have not faced any incidents or assistance requests of a significant nature (i.e. any issue that was not able to be satisfactorily resolved on the spot). This section however attempts to cover the issues that were encountered by first briefly recounting the technical glitches and then to discuss the reaction of voters and poll workers to Punchscan, and more broadly to E2E elections in general.

6.5.1 Technical Issues

There were a number of issues that the pollworkers experienced. For example, of the 154 ballots cast, only 145 were recorded in the electronic pollbook. This means that nine instances occurred in which ballots were cast without the voter's eligibility being verified. However this is a matter of pollworker procedural compliance, and therefore is no different than any existing voting system in this regard.

More interesting is how the pollworkers reacted (uncued) to technological failures. On the software end, there were several instances in which either the polling place or electronic pollbook software froze. On the hardware end, there were several instances in which the printers jammed or the wireless internet signal was


Welcome to Punchscan - covering the GSAÉD executive election
File Edit View History Bookmarks Tools Help
http://punchscan.org/gsaed/view_ballot.php?zoom=0.25&
Welcome to Punchscan - coverin...
VP INTERNAL / VP AUX AFFAIRES INTERNES
Kelly McClellan
Yes / Oui
No / Non
B
A
VP FINANCES / VP AUX FINANCES
Rizwana Alamgir
Yes / Oui
No / Non
A
B
VP SERVICES / VP AUX SERVICES
Faisal Deen Arif
Yes / Oui
B
A
students' rights to fair and
tion?
Êtes-vous d'accord pour
à temps plein par semestre
temps partiel par semestre
Centre de recours étudiant,
protégera et défendra le dro
une représentation juste et
Yes / Oui
No / Non
Directions:
1. Choose your candidate/selectio
2. Find the matching letter in one
3. Firmly and gently place your in
4. Repeat for each contest.
Done


VP INTERNAL / VP AUX AFFAIRES INTERNES
Kelly McClellan
Yes / Oui
No / Non
B
A
VP FINANCES / VP AUX FINANCES
Rizwana Alamgir
Yes / Oui
No / Non
A
B
VP SERVICES / VP AUX SERVICES
Faisal Deen Arif
students' rights to fair and
tion?
Êtes-vous d'accord pou
à temps plein par semestre
temps partiel par semestre
Centre de recours étudiant
protégera et défendra le dr
une représentation juste et
Yes / Oui
No / Non
Directions:
1. Choose your candidate/select
2. Find the matching letter in on

lost. During these instances we found that all the pollworkers undertook to create a handwritten account [3] of the cast ballot, interestingly in some cases without having been instructed how to do so. In the absence of the printer to create overlays and digital signature, the poll workers signed the ballots manually. These handwritten records were later recorded electronically, and transcription errors would be subject to detection through the online receipt verification check. This demonstrated an unexpected robustness of Punchscan against a host of denial of service scenarios.

6.5.2 Psychological Acceptability

The act of marking a Punchscan ballot by matching shuffled letters has been referred to as "indirection". Opinions of voters varied widely over the degree to which indirection was an issue. Some voters actually claimed that Punchscan was "no harder" to mark than a traditional ballot, whereas others found it irritating. Here the voters were more concerned with their personal experience than in their ability to correctly transcribe their voting intent on a Punchscan ballot. In order to gage the usability of Punchscan however, the rate of occurrence of intent transcription errors would have be measured. However in the context of voting systems, a proper user-study would require the ability to maintain a linkage between individual voters and their vote in order to gage their ability to successfully cast their vote as they had intended. Obviously this is not possible during a live secret-ballot election. However the reactions of the participants during this case-study do point to important questions a user study should undertake to answer. The design principal of psychological acceptability can be applied to voting technology [19], postulating that the security mechanisms of the system should be congruent with the voter's mental model of the system. Our case study confirmed that the security mechanisms of Punchscan were not well understood by the voters.

6.5.3 Ballot marking

The fact that the order of the letters on the ballots were randomized was not clearly indicated on the ballot, leaving the voter with only their intuition for forming a proper understanding of why a receipt does not contain adequate information for determining their vote. The randomized lettering was likely the predominant barrier in understanding. However once this was explained to the voters, many understood in a flash of insight. However understanding did not necessarily precipitate willingness and many voters indicated their sense of burden with the voting process. What was clear from the election was that voters did not intuitively understand the ballot marking process. Most became satisfied of the requirements after a verbal explanation. However the quality of explanations and (therefore their effectiveness) varied between the pollworkers. As of yet, there still is not an standard script on how best to educate new voters.

6.5.4 Shredding

Also at issue was the process of shredding one of the ballot sheets. Concern was expressed by several voters over what they perceived as the destruction of their vote. Many were also confused by being given the option to shred either page, in some cases asking several times to ensure they understood correctly. This might be attributed to the fact that typical administrative tasks (such as filling out a government form) do not offer options as to how the task should be completed. Therefore we might reasonably conclude the choice component of ballot completion is contrary to the voter's mental model. The original design intention was for voters to retain top and bottom sheets with near equal probabilities to reveal ballot tampering. Interestingly however given the choice approximately 85% of the voters chose to retain the bottom page as their receipt. On the third day of polling, after this trend had been established, the Punchscan team questioned voters leaving the polling station why they chose the sheet to shred that they did. The majority explained that because the ballot was still locked to the clipboard, they found that ripping off the top sheet to be the easiest way to complete the action. However in a subsequent Punchscan election, held at the 2007 Computers, Freedom and Privacy (CFP) conference, where clipboards were not used, approximatly 85% of the 36 voters still chose to keep the top sheet. The implication of receipt skew makes it less probable for attacks to be detected during the post-election audit. Although the degree of receipt skew in this election

[3] Serial number plus a numeric code for each mark (or absence of mark) made by the voter.

didn't significantly affect a reasonable assurance of integrity, it is something that should be better understood for future elections.

6.5.5 Conveying the Purpose of Punchscan

Another source of dissonance between the ideals of Punchscan and voters' mental model concerned the purpose of ballot receipts. Many did not realize they were going to receive a receipt and wanted to leave immediately after marking their ballots. Furthermore, when it was explained to them that would receive a receipt, and what the receipt allowed them to do with respect to independent verification, a number of voters were still indifferent. It is important to the integrity of the election that voters at least take the receipt—a left receipt means that receipt will not be checked, opening a window of opportunity for an attacker to modify that ballot.

Voters' reactions demonstrated that E2E has a long way to go in making its benefits clear to voters. Its not necessary for the voters to have a perfect mental model of the system. It is difficult to gage how fundamental these problems are to Punchscan, and E2E systems in general. They could be simply rooted in the novelty of this kind of system, in which case voter education would go a long way to resolving these issues. Our case study does indicate that as some voters became aware of the verification ability afforded to them with Punchscan, they were accepting of the extra measures required for voting.

The ballot receipts were hosted on the Punchscan webserver and the server logs show that the image files of 83 of the 154 ballots got at least one hit. Whether this means they were actually checked against the receipt cannot be determined from available data, but it is suggestive of an interest by voters in checking their ballot receipts online.

6.5.6 Poll Worker Feedback

The poll workers opinions varied as to the difficulty of the ballot casting procedure. All polling stations encountered minor computer glitches at some point during the election, as well as the occasional printer jam. The polling place software also experienced some buggy behavior after being left running for long periods. However recalling from earlier, on the occasions that there were technical complications, the poll workers were able to record the ballots manually. Their comments include the need for "a more detailed explanation of the voting process on the ballots." One poll worker explained to us that a voter daubed the serial number holes, which in turn caused a serial number recognition error at the polling station. Also echoing the comments of voters was the need for an explanation as to "why the voters have to destroy a page." They also suggested visual instructions posted inside the polling booth, a brochure explaining Punchscan's "security features."

Perhaps the most widespread concern however was the time to cast. Obviously 154 voters across 5 stations and 3 days allowed for the poll workers to take their time casting votes. However, as one poll worker described it, "I don't know how it will work if we have people lining up." We did not conduct stopwatch-timed trials during the actual election, however generally we found unlocking the clip board, scanning and casting the ballot, and then printing overlays took around 60 and 90 seconds. On the other hand, at the subsequent CFP election we used Punchscan in what we call "mail-in" mode, which in this context meant that instead of shredding one half of the receipt, the voter kept one half, and gave the other half to the poll worker. The collected receipts are scanned at a later date. Therefore the time to cast for this variant was the time it took to hand the poll worker one of the ballot sheets (effectively zero). However the mail-in version does not offer the same degree of privacy or integrity as the full-scale version used in the GSAÉD election.

6.6 Metrics

6.6.1 Average time to vote

The average time to vote varies according to the additional features—overlays and receipts—used. Figure 6.5 summarizes this information. We did not conduct thorough time tests during the election, and these statistics are approximations.

Version	Time to Cast	Integrity	Privacy
Clipboard, Scan, and Overlays	60-90 seconds	Very High	Very High
Scan and Overlays	40-70 seconds	High	High
Scan Only	20-40 seconds	Medium	High
Mail-in	~	Medium	Medium
Standard Paper Ballot	~	Low	High

Figure 6.5: Time to vote statistics. These statistics do not include time to mark a ballot, only the time added by the Punchscan architecture over-and-above a standard paper ballot.

6.6.2 Cost

The cost of a Punchscan election can be considered in terms of fixed and marginal costs. The fixed cost is the cost to cast the first ballot, and it includes all the overhead costs. The marginal cost is the cost to cast an additional ballot given that the fixed costs have been accounted for. We now consider these costs per polling place in Canadian dollars.

We estimate the cost of the equipment as follows: a basic computer capable of running JRE ($200), a flatbed scanner ($100), an ink jet printer ($200), a paper shredder ($50), a clipboard and lock ($20), an ink dauber ($1), a polling booth ($10), and ballot box for the paper backup receipts ($10).

The ballots must be drilled, printed, and then have overlays printed over them. At 19 holes per ballot and 6 reams of paper, the setup and drilling was completed in under half an hour. Given the machine and operator rate of $75/hour, 3000 Punchscan ballots were produced at a unit cost of $0.01. Using standard quality white 8.5x11" office paper costing $0.01 per sheet, the overall until cost of an unprinted Punchscan ballot was $0.03. Hewitt Packard lists the cost per page (CPP) of printing black text on the HP K550 Officejet Pro as $0.015. A Punchscan ballot is essentially black ink, with only small yellow alignment marks. The overlays are color however they are sparse and not near a full page of text. We estimate the CPP of printing the ballot and the overlays to be $0.03, bringing the unit cost of a ballot to $0.06.

Poll workers were paid $10 per hour and two poll workers were at each station. Average time to vote is 1.5 minutes, so wages for a vote are $0.51.

Totaling these costs together, the fixed cost to cast the first ballot is $591.57. The marginal cost to cast an additional ballot is $0.06 (or $0.57 with labour).

6.7 System Performance

In the end, this election was successful in the sense that its participants (the Punchscan team, GSAÉD, the voters) were sufficiently satisfied in Punchscan's ability to fairly and accurately conduct this election. The elected candidates became ratified by the council, and no challenges were made against the election results. The election was modest and not hotly contested, but it is our position that this case study represents an important milestone for Punchscan and E2E elections in general.

Chapter 7

Known Issues and Failure Modes

7.1 Bugs and Known Issues

In the context of voting systems, the need for transparency of the election machinery is arguably as high as any other environment where software is used. In this situation, open source software provides disclosure of the source code for the purpose of independent review to promote the identification of security vulnerabilities. Although the security of an E2E voting system rests primarily on the cryptography, the correctness of the implementation (i.e. code) does play a role in addressing the potential for attack through side-channels. More fundamentally however, it is essential that the user has a reasonable assurance that when, for example, the audit tool responds with positive verification of the election data, it actually carried out the cryptographic checks.

The Punchscan source code was released to the public on November 2, 2006 under a modified BSD license. However despite this, there has not been a thorough review of the code to date by an independent source.

In the GSAÉD election, several bugs arose. The polling place software would occasionally freeze although the source of this problem as not been identified.

There are also components of the software which function correctly but are highly sensitive to proper execution; namely the color calibration in the polling place software. The calibration requires the poll workers to identify and click on the ink daubs on a scanned image of a marked ballot. This process documents the mean and variance of color that should correspond to a mark. Failing to be attentive when clicking on marked regions or selecting an insufficient range of colors could cause the OCR to not recognize marks in scans. Since the recognized marks are shown to the voter at the polling booth and printed over the ballot receipt as overlays, scanner errors should be caught before the voter leaves. A misconfigured image file can then be rectified through a more attentive reconfiguration.

7.2 Missing Parts

The Punchscan system has the potential for extensive support for persons with disabilities. For example, if a voter is unable to see and unable to physically mark her ballot, audio versions of the ballot layers may be employed. The voter can choose which ballot layer she will retain as a receipt before entering the booth, and have the other layer immediately shredded. The chosen layer can be given to an aide, and the voter enters the booth with an audio version of both layers. On hearing the audio layers, and determining which disc is to be marked, the voter can request the aide to mark the layer accordingly. Given that the aide has only one layer, he does not know what vote is being cast, only which disc is to be marked. Further, the voter may record her requests and take this tape (digitally signed by a computer) home with the paper receipt. She may check that her paper receipt is online by listening to an audio version of it in order to check that the voting system has counted her vote as cast by the aide and that the aide cast her vote as instructed. Support for audio versions of the ballot layers and for audio versions of the online receipts has not yet been implemented in the current version of Punchscan.

There is one technical piece of the system that is missing. The website does not yet support changes we

made to the XML schema that supports certain voting rules. This is more a problem of miscommunication rather than a technical problem and should be fixed by the beginning of the competition.

7.3 Failure Modes

The Punchscan system is extremely robust and can operate when components fail. For example, recall that the polling place uses a computer, scanner, and printer. If the printer fails—due to an ink shortage or a paper jam—the overlays can be hand written by the poll workers and the receipt can be signed by the poll worker. The paper receipt can be generated by hand as well. If the scanner or computer fails to work, as in the case of a power outage, the voter can simply give place one half of the ballot in a ballot box (the half that would have been shredded) and keep the other half as their receipt. When the scanner is operational again, the ballots in the box can be scanned by the poll workers. Similarly if the electronic pollbook book fails, due to a network failure, the names of voters can taken by hand and the lists of each polling place can be compared throughout the day to combat double voting. In the event of catastrophic data loss, the surviving paper receipts can be scanned or transcribed (i.e. typed) back into electronic form.

The existence of errors in the scanning or tallying will be detected, with a probabilistic certainty, through the pre- and post-election audits in combination with voters checking their receipts. If the tallying fails the audit, it can be rerun. If a posted ballot receipt differs from a voter's physical receipt, she may be return with her receipts to rectify the situation.

The election data needed to reconstruct an election is fully deterministic with the shared secret key. For this reason, none of the election tables need to retained in any form other than a threshold of trustees remembering their passphrases.

Chapter 8

Security Analysis

Proper security analysis is an arduous effort. We believe the thorough analysis that a voting system deserves, particularly one such as ours, requires more resources and expertise then we can provide. Even for systems that appear to be relatively less complex, such as Direct Recording Electronic (DRE) and Precinct Count Optical Scan systems (PCOS), thorough security analysis is an extremely difficult task as evidenced by, for example, the Brennan Center report [2]. Despite this, we can make concrete claims on a variety of known threats.

8.1 Trust Assumptions and Threat Model

In general, we model many of our trust assumptions and threat models around what can be expected in a precinct voting situation. To this end, we first assume that the election authority is comprised of a committee with opposing interests (*i.e.,* 2 Democrats, 2 Republicans, and a chair). We further assume that polling places will work similar to today's poll sites—being run by volunteers from different parties and a poll captain. The competing interests of the EA and the poll-workers allow us to assume they will be non-colluding. We also assume the existence of an effective, secure method for voter registration to prevent double-voting[1]. Finally we assume a secure chain of command to ensure that printed ballots are sealed after printing and delivered to the polling place without the information on them having been seen or copied by anyone.

As been stated, our system provides unconditional integrity and computational privacy. You might ask why we choose integrity to be paramount instead of privacy. Our philosophy is that integrity is more important because without it, it is impossible to detect a corrupt election. Without privacy, voters are at least aware of the infringement if the ruling body were to act on the information and retaliate against voters. That said, the privacy offered by Punchscan is still extremely robust and may even surpass anonymous paper ballots in certain regards (e.g. voters leave fingerprints on ballots, whereas in Punchscan, only an electronic copy of the ballot receipt is retained). Unconditional privacy is virtually impossible to enforce with the existence of sophisticated surveillance equipment that could be employed by a spying election authority or by voters themselves wanting to purposely infringe on their own privacy to sell their vote.

Our threat model considers what voters, poll workers, members of the election authority, and election observers can do individually to thwart the integrity or privacy of a Punchscan election. We also consider what could be achieved through collusion between members of the same interest group, but consistent with our trust assumptions, we do not consider collusion across all interest groups.

8.2 Security Claims

Punchscan is a cryptographic voting system, not an electronic voting system. The security of Punchscan relies in mathematics, not in the ability of a computer to keep sensitive data secure. Computers are primarily used to increase the efficiency of the calculations. A Punchscan election uses three distinct classes of computers:

[1] We believe there are some favorable properties to a reliable record of who has voted in an election. If you can segregate bad votes from the good, you can pull them out of the count.

the computer used to run the election meetings, the computers used at the polling places, and the server that run the website.

The polling place computers and webserver only come in contact with publicly known election information. The polling place computers make digital copies of the receipts and the webserver provides these to the public, along with the election results and audit data. The security requirements for these computers are not strong, although we will consider a few threats relating to them.

The only secure computations required by Punchscan are computed during the meetings. In this case, the election officials enter their passphrases into a computer and all the election data is deterministically generated from the passphrases. The internal state of the computer during these computations must be protected. In order to facilitate this, the Punchscan software can be executed from a bootable USB stick. The source code for the software is public, and a hash of the code is published as well. Before the meetings, each election official will bring a copy of the software and they can check the integrity of the other officials software by computing a hash of it and comparing it to the publicly posted hash. Once convinced, a randomly selected copy of the software can be used to boot an opened computer without a harddrive, any form of permanent memory, or a network connection. This ensures the software only executes in RAM by properly formed software, and that the data is lost when the computer is disconnected from its power source.

We assert that these measures give us a reasonable assurance that the election key will not be recovered by a hacker.

8.2.1 Breaking the Underlying Cryptography

Punchscan relies primarily on two cryptographic algorithms: the SHA-256 hash function and the AES block cipher. It is theoretically possible that unknown cryptanalysis may render these functions insecure and allow the election key to be recovered, and that a government agency has the capability to accomplish this today. If this were the case, the privacy of the election would be compromised and votes could be linked to serial numbers. However the integrity of the election would still be preserved with a high degree of probability. The best an attacker could do, besides tampering with information and hoping to avoid detection (which does not require a breach of the cryptography and will be discussed subsequently), would be to find a way to change a piece of election data without changing the commitment. In other words, if the attacker could easily find hash-collisions (two data streams that hash to the same value), she may hope to tamper with a ballot while avoiding the trail that her tampering would normally leave.

This line of attack has a low probability of working, even for a computationally unbounded attacker. Recall the committment algorithm from section 3.6.2:

Algorithm 8: Commitment

Input: MX_i, Skm_i, C
Output: CX_i

1 $Sak_i = \texttt{AES128}_{Skm_i}(C)$
2 $h1_i = \texttt{SHA256}(MX_i || Sak_i)$
3 $h2_i = \texttt{SHA256}(MX_i || \texttt{AES128}_{Sak_i}(h1_i))$
4 $\mathrm{CX}_i = h1_i || h2_i$

The attacker would change the value of MX_i (the data being committed to) and then attempt to find a suitable Sak_i (salt) such that the value of $h1_i$ is preserved. Working backwards, she can then find a suitable Skm_i to publish, assuming AES is broken as well (the correct derivation of Skm_i from the master key is not checked, as that would require knowing the master key, and so Skm_i is simply published by the election authority). Since Sak_i must be 128-bits, it is highly improbable that a collision even exists. However if one does exist, the probability that $h2_i$ would compute to the same value with the modified Sak_i and MX_i is negligible.

8.2.2 Tampering with Virtual Ballots

We have seen that an attacker is unable to modify election data without creating discrepancies between the data and the commitments.[2] However, since not every commitment can be checked against the original data, the attacker might risk tampering with the data and hope to evade the checks and balances Punchscan employs to discover tampering. The first opportunity for tampering would be hacking into the published commitments and changing them (or simply mispublishing the commitments). There is very little strategic benefit to doing this, since an attacker's goal is to rig the election for a candidate. Tampering with election data before the election is even run is not likely to achieve this goal. In any case, the pre-election audit looks at half of the commitments and checks their integrity.

This audit procedure ensures that the ballots are well-formed, meaning that for each ballot, $P_1 + P_2 = D_2 + D_4$ for the row in each D-matrix associated with that ballot. Suppose there are n ballots, the election authority has cheated by malforming k of them, and f ballots are chosen at random to be examined. The probability that the election authority gets away with this cheat is the number of possibilities where the auditor chooses only valid votes divided by all the possible choices. The number of all the possible choices is n choose f. The number of ways to choose f all valid votes from a total of n where k of then are invalid, is $\binom{f}{n-k}$.

Thus the election authority cheats and gets away with it with a probability:

$$p = \frac{\binom{f}{n-k}}{\binom{f}{n}} = \frac{\frac{(n-k)!}{f!(n-k-f)!}}{\frac{n!}{f!(n-f)!}} = \frac{\frac{(n-k)!}{(n-k-f)!}}{\frac{n!}{(n-f)!}}$$

Note that $f + k < n$, so that $n - k - f > 0$ and $(n - k - f)!$ exists and it isn't in the special case of 0!. If $f + k > n$ the probability is 0.

In the interest of simplicity, from here we may compute two upper bounds on the chance that this attack will not be detected:

1. $\frac{(n-k)!}{n!} \times \frac{(n-f)!}{(n-k-f)!} = \frac{1}{n\times(n-1)\times...\times(n-k+1)} \times (n-f) \times (n-f-1) \times ...(n-f-k+1) = \frac{n-f}{n} \times \frac{n-f-1}{n-1} \times ... \times \frac{n-f-k+1}{n-k+1} = (1-\frac{f}{n}) \times (1-\frac{f}{n-1}) \times ... \times (1-\frac{f}{n-k+1}) < (1-\frac{f}{n})^k$
2. $\frac{(n-k)!}{(n-k-f)!} \times \frac{(n-f)!}{n!} = (n-k) \times (n-k-1) \times ... \times (n-k-f+1) \times \frac{1}{n\times(n-1)\times...\times(n-f+1)} = \frac{n-k}{n} \times \frac{n-k-1}{n-1} \times ... \times \frac{n-k-f+1}{n-f+1} = (1-\frac{k}{n}) \times (1-\frac{k}{n-1}) \times ... \times (1-\frac{k}{n-f+1}) < (1-\frac{k}{n})^f$

Thus, our upper bound on the probability that the Election Authority gets away with malforming k out of n ballots when f of those ballots are audited is $min[(1-\frac{f}{n})^k, (1-\frac{k}{n})^f]$. We assert that when $f = n/2$, any significant tampering will be caught.

8.2.3 Misprinting Ballots

The election authority may also misprint ballots. This is counter to the non-collusion assumption we have made, but even if this were to occur, random ballots may be checked to see that the printing matches the permutations dictated by the ballot authoring process. Further more, half of each ballot will be kept by the voter as a receipt and can be checked after the election to see that the permutations on that half are properly formed. In order to check that a given ballot receipt was properly printed, one can reencrypt it (that is, recompute the commitments) and compare it with the P-matrix. Suppose n ballots remain unspoiled after the pre-election audit, f are actually used by voters who later check the commitments, and k of them are improperly printed. Once again, the upper bound on the probability that none of the misprinted ballots are detected is $min[(1-\frac{f}{n})^k, (1-\frac{k}{n})^f]$.

8.2.4 Tampering with Tallying Inputs

After the election is complete, an attacker may modify the the inputs to the tallying function is order to swing the results toward a favored candidate. This could be accomplished by hacking the polling place

[2] This section taken primarily from [16]

computers to misrecord the scanned ballots (although at this point, the results of the election would be unknown and since the hacker does not know the vote on the ballot, she may inadvertently be taken votes away from her candidate of choice). The inputs the tallying function are the ballot receipts, every one of which may potentially be checked by a voter. This attack violates the non-collusion principal of the poll workers, but even still has a high probability of being caught by voters checking their receipts. Again, if n ballots remain unspoiled after the First Audit, f are actually used by voters who verify that their ballot marks are correctly recorded online, and k ballots are incorrectly recorded, then the upper bound on the probability that none of the incorrectly-recorded marks are detected is $min[(1-\frac{f}{n})^k, (1-\frac{k}{n})^f]$.

8.2.5 Tampering with Tallying Data

The attacker may also tamper with the election data contained in the D-table. This possibility is excluded by the assumption that the software that generates the data from the master key is correct and run on a secure computer. However even if it were to be possible, the tampering would have to be consistently applied to each independent D-table so that the results are consistent across the tables. This alone is difficult to achieve since the the rows in the D-table are randomly shuffled with a different shuffle for each table. Furthermore, the tampering would have to hid in either the left or the right side of each table. Note that tampering with the output of the tallying function is equivalent to tampering with the right-side of the D-table. The post-election audit opens one half of each table, and so the attacker would have to correctly hid all of the tampering in the side that is not opened in each table. The probability of getting caught is $1-\frac{1}{2^d}$ for d independent D-tables. For example, with 10 D-tables, the probability of catching such an attack with the post-election audit is over 99.9%.

8.2.6 Subverting the Audit

The success of the pre and post-election audits assume that which ballots and which D-table halves will be audited are unknown to the attacker.[3] If the attacker could arrange what data is to be audited ahead of time, she could successfully hid the tampering in data she knows will not be audited. It is thus essential that the audits be random and unpredictable to the attacker. The best way of achieving this is to use a random entropy pool such as dice or lottery balls. However these types of measures are not voter-verifiable unless if the voters are in the room when the dice are thrown or the balls are drawn. What is needed for this process is random data that is easy accessible to any voter or election observer. For this reason, Punchscan uses stock data in the fashion outlined in Section 3.8.1.

Although it only takes a single stock for the output to be unpredictable, an attacker can work with less than complete unpredictability to compromise the random audit. Consider the situation where a single stock is used. The one bit of entropy produces one of two psuedorandom streams: say, `100110...` or `010010...` We can assume that the attacker will not be able to predict which of the two streams will be produced. However the fifth bit in both selection streams is a 1. This means the fifth ballot will not be audited regardless of which stream is generated, and so this ballot could be safely corrupted even though the attacker cannot predict the value of the seed.

We define a *corruptible ballot* (CB) as a ballot that will not be selected for auditing for all possible values of the seed. Finding corruptible ballots will be referred to as an *intersection attack*. This attack requires the adversary to generate the output stream for every possible initial seed—an N-bit seed produces $S = 2^N$ streams—and calculate the intersection (bitwise AND) of all the streams. For every 1 in this intersection, the corresponding ballot is corruptible. Assuming the output of the pseudorandom number generator is uniformly random and statistically independent for each different N-bit seed, the probability that a given ballot b is a corruptible ballot is:

$$\Pr[b \text{ is CB}] = \frac{1}{2^S} = \frac{1}{2^{2^N}} \tag{8.1}$$

We can take two approaches to defeating the intersection attack. The first is to ensure it is computationally infeasible to run every possible seed by making the seed space sufficiently large (to the order of 128 bits). A second option is to make S merely large enough that the probability of encountering even one

[3] This section taken primarily from [7]

corruptible ballot in a B-ballot election is less than a half. Rearranging the above equation, this criteria requires $N \geq \lceil \log_2 \lceil \log_2(2B) \rceil \rceil$. Recall the voting age population of the United States was 215,694,000 in 2004 and that B is twice this (because the half of the ballots that are audited are thrown out). This would require a portfolio size of $N \geq 5$ to defeat the intersection attack. Using this minimal N, the expected number of corruptible ballots in an election of this size is 0.1.

However we should also consider ballots that have a low but non-zero risk of being audited. For example, a ballot may be selected for auditing in only 1 of the S streams for each possible seed. For a large S, the probability that the one seed that would select this ballot would be generated is quite small. We define *risk level*, R, to be the probability that a given ballot position will be audited across all the seeds. A corruptible ballot has a risk level of 0, while a ballot that will only be audited in say 1 out of 16 selection streams has $R = 0.0625$. In theory, each ballot should have a 50% chance ($R = 0.5$) of being audited. We define a *semi-corruptible ballot* (SCB) as a ballot that has a risk level R, for any $R < 0.5$. The probability that a ballot is a semi-corruptable ballot with risk level R or less is,

$$\Pr[b \text{ is SCB}_R] = \sum_{i=0}^{\lfloor 2^N \cdot R \rfloor} \binom{2^N}{i} \frac{1}{2^{2^N}}. \tag{8.2}$$

The following table shows the minimal portfolio size required for different sized elections to ensure the expected number of semi-corruptible ballots with a risk level of 45% is less than one half. Thus, for an American election, the minimal portfolio size is 12 stocks. We recommend exceeding this number by a safety margin and thus suggest using a 20 stock portfolio.

Portfolio Size	Number of Ballots
8	0 – 8
9	9 – 41
10	42 – 783
11	784 – 172,382
12	172,383 – 6,262,358,931

Table 8.1:

In any stochastic system, the passage of time increases the uncertainty of predicting the final state of the system. The famous formulation of the butterfly effect posits that a butterfly flapping its wings in Brazil may induce a hurricane in Texas many months later. The point is that it takes months for the two weather systems (one with the flapping, one without) to diverge enough for their difference to be as monumental as a hurricane. In our stock market model for randomness, the required difference is much less monumental—a couple of cents is sufficient.

The election authority publicly commits to using the closing prices of a set of stocks at some future time. For the resulting random number to be unpredictable, we need to allow the market adequate time to probabilistically diverge from its current price by at least a cent. There is limited consensus on the underlying statistical mechanics of the stock market, and so it is impossible to exactly determine the minimum amount of time required. However we have the benefit of erring on the side of caution, so we will approach this problem by asserting that one day is more than adequate and then verifying this assertion statistically.

The volatility of the market is typically calculated in terms of the rate of return,

$$r = \ln \frac{P_{t+1}}{P_t}. \tag{8.3}$$

P_t is the price at time t, while P_{t+1} is the price at the next time period. The return is logarithmic because stock investments represent continuously compounded interest. Volatility is typically calculated using the standard deviation of a stock's returns, σ_r, over a period of many years and then annualized,

$$\sigma = \sigma_r * T^{\frac{1}{\alpha}}. \tag{8.4}$$

T, the time division, is 12 or 252 if monthly or daily rate of returns are used respectively in calculating σ_r. α is the divergence factor and a value of 2 is typically used, which assumes that rate of returns follows a

random walk. This assumption is widely held but controversial [13]. Some financial analysts believe it is less volatile [12], while other mathematicians such as Benoit Mandelbrot posit that the market is scale-invariant and thus advocate smaller alpha values [14]. The value of σ for a given stock is rarely published; instead volatility is given by β,

$$\beta = \frac{\text{cov}(r_s, r_m)}{\text{var}(r_m)} = \frac{\text{cov}(r_s, r_m)}{\sigma_m^2}. \tag{8.5}$$

β measures the volatility of returns of a given stock (r_s) against that of the market (r_m). A $\beta > 1$ means the stock is more volatile than the market, while a $\beta < 1$ means it is less volatile. By convention, β uses the daily returns for the past 5 years, and the "market" is the performance of the S&P 500.

While readily available, β it is not the best metric for our purposes. We are interested in simple stock movements not returns (e.g. a stock moving up or down 10% is equivalent to us, whereas in terms of continuously compounded returns, the movement down is worst). We can define the movement of a stock, m, to be the absolute value of an arithmetic return of investment, $m = |(P_{t+1} - P_t)/P_t|$, and calculate it daily for the S&P 500 over the past 5 years (ending December 29, 2006). Using the actual market data for this period, we determined that the index had a mean value of \$1120.67, and a low/high of \$776.76 - 1427.08. The expected value of the daily movement was .73% or \$7.69 with a standard deviation of .71% or \$6.71.

This movement calculation suggests we should only choose stocks valued over \$10 to ensure a reasonable certainty that the stock will move by at least a cent, and that average volatility of the portfolio be as volatile as the S&P 500. To ensure that it is, this calculation could be performed on each individual candidate stock for five years of data. However given that the β value is published and readily available, we will use it instead and recommend that a portfolio with a $\beta > 1$ is chosen.

Given that a properly sized portfolio is chosen, that it is sufficiently volatile, and that the audit day is announced at least 24 hours in advance, we assert that the random selections for the audits are unpredictable to an attacker and cannot be subverted.

8.2.7 Chain voting

Punchscan avoids this whole class of problems by using its own innovative technique of issuing voters that sign-in with large, uniquely-colored clipboards that physically lock ballots and receipt sheet choice. Clipboards are unlocked only when ballots are cast and an unlocked clipboard is handed over before exiting the polling place. Chain voting in this instance would require a very fast lockpicker and/or the cooperation of poll workers watching the exit while avoiding polling place observers.

8.2.8 Denial of Service Attack

With punchscan, even destroying all the electronic and centralized physical records, at least in principle, does not prevent the outcome from being re-constructed using receipts held by voters. The keys needed to complete the election and audit are shared according to rules, which can be set up to ensure that there are combinations of authorities sufficiently likely to allow the election to complete.

8.2.9 Fake Receipts

This would require counterfeiting of receipts at some level and probably would not be the easiest way to disrupt an election. In paper-based elections, used around the world, the appearance of ballots outside polling places casts doubt on election integrity. But anti-counterfeiting techniques developed to protect currency can be very effective, such as by hidden taggants in paper and ink. So other forms of disruption, such as deliberate but deniable errors in setting up elections, malicious software later revealed, and discoverable tampering with data can be less costly and longer lasting. Moreover, current systems are essentially defenseless against the growing claims that they offer little actual protection and they lose all credibility if the multiple records they maintain ever become inconsistent.

8.2.10 Random Voting

If a vote-buyer or coercer tells the voter where to mark before the voter enters the polling place, then those marks will correspond to random votes. This is because only in the booth can the voter see what vote corresponds to what mark on the receipt. So paying for such marks is actually like paying for random votes, which is substantially the same as paying someone not to vote at all. But paying people not to vote can be achieved more directly (and even online), since who votes in US elections is generally visible and in practice a matter of public record. Influencing voters not to vote on certain contests, while allowing them to vote their choice on others, is essentially a waste of the influencer's efforts, which would generally be more effective if the voter were kept from voting altogether[4].

Observing how long people spend in booths has also been used in improper influence schemes, and lever machines even make a distinctive sound for each contest voted. Nevertheless, an overvote position (inherent in many other paper-based voting systems) combined with a mark per contest requirement, lets a punchscan receipt hide even which contests were voted.

There is a conceivable attack where voter's are told which sheet to take after they have seen a ballot, and this would give an attacker a higher chance of success. However, in our specifications, the receipt choice is made before the voter knows the contents of the ballot.

8.2.11 The Italian Attack

In the italian attack, voters are given a unique set of votes and undervotes to record on their ballot. After the election, an attacker looks for this unique fingerprint in the results table, and rewards or punishes the voter accordingly if he does or does not find it. This attack is only possible with a large number of choices are being made, such that there is an extremely large number of possibilities, and the attacker picks those which are not likely to be present after the election.

We thwart this attack by decoupling the races from each other, using a separate Punchboard for each race. This process is detailed in [17]. The only caveat is that the attack may still work in races where there are a large number of candidates and the voting rules force voters to rank each candidate.

8.3 Conspiracy Resistance

As discussed in section 8.2.1, for all practical purposes it is impossible to undetectably affect the count. In that sense the tally can never be incorrect (whatever is on the website/audited is always counted correctly), but the election might still be rigged in one of the following ways:

1. Conspiracy between the Auditors and EA.
2. EA Conspiracy to avoid registration.
3. Coercion Attack.
4. Take advantage of lost receipts.

For 1, the EA will tell the Auditors what selections to choose, and it can modify votes at will. However, this is an unlikely scenario as we think the Auditors should be the candidates themselves. What motivation woud you have as a candidate to help your competitor win?

For 2, the EA could create a shadow pollsite or shadow voters that vote for the candidate of their choice. Since what goes into the tally is public, it is likely that such an attack would be caught. An additional protection we could add is a signature of each receipt on the website signed by a local polling site authority. That way the receipts could be checked to see if they came from legitimate polling sites. In any case, this is typically a large endeavor, and is unlikely to work or go undetected.

For 3, the EA itself, or an Attacker who has broken the computational protections on the privacy, or a combination thereof carry out a large scale coercion attack on voters. The problem with such attacks is that

[4]There are situations in which random voting may be desireable, such as trying to avoid suspicion when forcing a larger portion of a population not to vote.

every coerced voter knows that they happened. It is unclear how long such an attack would go on without notice, but we imagine that it would not be very long.

For 4 to work, an attacker or the EA would have to know of a number of receipts that will not be checked by voters or a 3rd party before results are posted. Then change the receipts online before the EA meets to calculate the totals. In order to do this in a meaningful way, they would also need to have broken the computational protections to violate the privacy.

As you can see, none of these attacks are really possible without cooperation of a large number of people, and this is particularly true if the computational protections actually work. In most cases, this can be assumed to be the case.

Chapter 9

Other Claims

This section details various claims and design decisions that did not fit very well anywhere else in this submission document. Please be aware that each section is independent of the others. We limit discussion in this section to points we feel are important to mention. We encourage readers to additionally read the FAQ on our website if they have questions that have not been answered in this document.

9.1 Scanning Advantages

We consider our scanning system advantageous for the following reasons:

1. **Marking Simplicity.** There is no special marker or marking process to vote with our system. Other systems we have seen require written character recognition, drawing a line across a page, or filling in a bubble, being careful not to mark outside the lines. Using the dauber and just pushing for the choice is easy to do, even for individuals without great dexterity.

2. **Interpretation Feedback.** The scanner itself, connected to a computer with a screen, gives positive feedback that voter's marks were properly interpreted. Instead of a small black and white ballot counter on a scanner system, voters and poll workers are able to check that the system properly interpreted the ballot marks.

3. **Human Readability.** Some systems accomplish similar security properties but at the cost of a hash or digital signature on the ballot. With the exception of a printed barcode that is only used in dispute resolution, everything on the ballot can be interpreted by a human. To do this did require optical character recognition for the serial number, but since we control it we can pick a font that is friendlier to machine interpretation.

4. **Digital over Analog.** With a digital scanner, while we do gain the (bad) capability to interpret identifying marks on the ballot, we also gain the ability to do digital manipulation and algorithmically improve scanning. It is also our belief that we could make the scanning process much faster using digital camera technology as opposed to flat bed scanners. Doing so would allow us to take photos of ballots and display results in less than a second.

9.2 Punchboard v. Tellers

The earlier, related system that Chaum proposed [4] uses a series of tellers. Punchscan could also utilize the teller model, the advantage being a physical separation of election authorities when generating and counting the ballots. While this model does not affect integrity, it is intended to increase the privacy protection in the system. The dominant reason that we chose not to use tellers is that the system uses printed ballots. When your privacy relies on printing with random bits on ballots you create a point of privacy failure in the system, because you have to give the data to print the ballots to the printer. Since this data comes from an election authority and goes to a printer, it can be argued that having multiple tellers defeats the purpose.

There are additional reasons:

1. **Speed.** The Punchboard is similar in intent to having only 1 teller. So, 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.

2. **Communications Requirements.** The tellers need to communicate with each other, and, depending on implementation, possibly with the polling places. In practice this would likely require an internet connection. The Punchscan system is off-line in the sense that data is hand-carried to a special room, done on a closed system, and then hand-carried back out. While you could do the same thing, all the overhead in the software validation we've added might make having to move to separate rooms undesireable.

3. **Trusted Hardware/Software.** Having multiple tellers does help with the trusted hardware and software problem, but it is not as effective as we would hope for. 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 homogenous. If an attacker can simply do the same thing to all tellers, it doesn't exactly have the desired effect of greatly increasing cost.

9.3 Indirection

Indirection is a term applied to the characteristic of a Punchscan ballot. Recall there are two sets of letters which are both randomly ordered. To vote, a voter must match the letter beside their chosen candidate's name to the corresponding letter showing through the holes (and then mark it). There has been some debate over the degree to which indirection impacts the voting experience. Specifically of interest are the following questions:

- How "frustrating" do voters find indirection?
- How does indirection affect accuracy?

It should be stressed that these remain open questions. However through our experience have received indications that would certainly form the basis of proper usability studies. With respect to the degree of "frustration" the voters experience, our expectation has long been that indirection essentially no harder than transcribing an answer onto an generic optical scan (e.g. Scantron) st5(t)-otictof the1.955Td[(than)-44aare

Obviously in the scope of a secret ballot election, the second item is not possible to precisely determine. We believe that a larger, well-funded study is necessary to explore this issue properly, but we have up to this point lacked the resources (i.e. money) and expertise to carry one out. User studies are tricky to conduct properly, and even if our approach was agreed to be sound, the findings would inevitably carry less weight unless the study is conducted by an independent body. Nevertheless, we've been exploring the possibility of providing independent verification without the use of indirection or randomized candidate lists. We've developed a solution called "Scantegrity."

9.4 Scantegrity

Scantegrity is an integrity assurance add-on for any conventional optical-scan voting system. It gives voters the ability to verify that their votes are recorded and tallied correctly without altering the basic form of the ballot or how voters use it. Instead of being a complete voting system, Scantegrity aims to provide a low-footprint audit solution "add on" to any current optical scan voting system. Its special audit symbols overlay on the ballot unobtrusively out of the way of what voters need to do-vote.

Figure 9.1: Scantegrity ballot depicting a generic ballot form with audit overlays (left) and serial/barcode chit (top-right)

Scantegrity uses a reduced version of the Punchboard. Since is shares alot of its clockwork with Punchscan, we have implemented it as an option in the Punchscan election engine. We will (optionally) demonstrate Scantegrity at Vocomp.

Chapter 10

Self-Evaluation

As per the requirements of 2.1.10 in the Vocomp 2007 rules [20], in this section we offer a self-examination of the Punchscan system architecture, our implementation of it, and our documentation of the system. In summary, Figure 10.1 shows our self-accessed scores. In the sections to follow, we justify our rational for these scores.

	Architecture	Implementation	Documentation
Cost	5	5	5
Voter usability and casting accuracy	4.5	5	5
Voter accessibility	5	2	5
Ease of administration	4	3	5
Reliability	5	5	5
Functional completeness	4.5	5	5
Integrity and verifiability of results	5	5	5
Ballot secrecy	5	4.5	5
Assured operations	5	5	5
Overall quality	5	5	5

Figure 10.1: Scores for self-evaluation.

10.1 Cost

The Punchscan system is extremely cost efficient for an automated voting system. The software can be used royalty free for student elections and has been implemented using open source development tools. Optical scan technology is much cheaper than touchscreen panels, and the Punchscan system can use any off the shelf scanner. Furthermore, Punchscan only requires one computer per polling place whereas DRE elections require a computer for every voting booth. These computers do not need to be cutting edge—essentially any computer capable of running JRE is sufficient.

For the GSAÉD election, we were able to keep costs down by borrowing several computers to use at the polling places and renting the rest. Since the election information scanned into the polling place computers is not secret information, there is no concern with a borrowed computer retaining election information. The election webserver and electronic pollbook was hosted on a school server, eliminating any server-side costs.

We also arranged for Hewitt-Packard to donate several printers for use during the election.

10.2 Usability and Acccessiblity

10.2.1 Voter Usability and Ballot Casting Accuracy

Punchscan requires the voter to match a candidate to a letter and then find the corresponding letter in the

the poll workers could retain the ballot layer that would have been shredded while the voter keeps the other half. When the equipment is functioning, the scanned ballot is displayed on a screen and the voter can verify it was scanned correctly. While this verification can also be done later by viewing the webpage, it allows the voter to catch any scanner errors while they are still at the polling place, and the receipt can be rescanned as necessary. If an error were to occur during the tally, the post-election audit would catch it with a near-certain probability. Furthermore, new decryption tables could be constructed after the election to use for tallying. These audit procedures are implemented in the current system.

10.5 Functional Completeness

Aside from the support for persons with disabilities which was accounted for above, the only function that cannot easily be supported by Punchscan is write-ins. Otherwise the architecture is very versatile and can accomidate any style of voting. This implementation was created for the GSAÉD election, which did not require write-ins and used first past the post (FPTP) tallying. Thus the implementation was functionally complete for the election.

10.6 Security, Privacy, and Availability

10.6.1 Integrity of Results and Verifiability of Results

As detailed thoroughly in Chapter 8 of this document, the Punchscan architecture offers high integrity elections through cryptographic principals and protects all votes from even a corrupt election authority through voters checking their receipts and the post-election audit. These security measures are currently implemented and the underlying cryptographic primitives in current use are AES and SHA-256 which are thought to be secure. The software is open source and open to inspection by security experts.

10.6.2 Ballot Secrecy

The Punchscan architecture can also provide a high level of voter privacy. Unlike in optical scan systems, the automation does not see how a vote was cast in Punchscan. To counter the authority trust problem, the election master key can be distributed among multiple trustees with competing interests (such as members of different political parties). The ballots themselves require a chain of command to keep the information secret but measures could be taken to seal the ballots in tamper-proof envelopes as they are printed. In the GSAÉD election, the ballots were simply printed under the supervision of the chief electoral officer who placed the printed ballots into sealed boxes and signed along the seal. Even though our implementation allows the use of threshold key sharing, GSAÉD chose to let the CRO be the sole trustee for the sake of simplicity.

10.6.3 Assured Operations

The use of audits in Punchscan allow us to assure that certain operations can be performed without having to rerun the election. The pre-election audit is designed to catch any software failure in creating the virtual ballots. As the ballots are being printed, random ballots can be inspected to ensure they match the virtual ballots (after which they are discarded). These two audits are not necessary to ensure the integrity of the election—the same errors would be caught later through voters checking their receipts and the post-election audit—however they do ensure that election will not have to be rerun. If the post-election fails, the tallying can be rerun with new d-tables. For this reason, once the election begins, a sound result is ensured. The only way to prevent a result would be for a plurality of trustees to refuse to enter their passcodes when the results are to be tallied—however the refusal of the election officials to proceed with the tally is possible in nearly all voting systems. The auditing software has been implemented.

10.7 Overall Quality

Through superior security and privacy protections, we feel that Punchscan is a high-quality, low-cost election system. As an E2E system, Punchscan offers voter-verifiability, an end-to-end audit mechanism, and ballot receipts which allow voters to see their vote count. Punchscan can be extended with audio and Braille ballots to not only make it accessible to disabled voters, but to preserve the secrecy of their ballots even when aided by another individual. Punchscan can be run on off-the-shelf equipment and a Punchscan election can proceed even if all the electronic components were to fail making it versatile and robust. Finally, the system is open source and is open to analysis and improvement by the computer programming community.

Chapter 11

Acknowledgements

We acknowledge our helpful advisors whom we consulted on this submission to VoComp:

- Carlisle M. Adams
  School of Information Technology and Engineering (SITE)
  University of Ottawa
  Ottawa, Ontario
- David Chaum
  Votegrity
- Jonathan R. Stanton
  Department of Computer Science
  George Washington University
  Washington, DC
- Poorvi Vora
  Department of Computer Science
  George Washington University
  Washington, DC

Although our team is composed only of those who have implemented parts of the submitted system, we would still like to recognize the other researchers we have worked with on topics related to the system:

- Kevin Fisher
  Cyber Defense Lab
  Center for Information Security and Insurance
  Department of Computer Science and Electrical Engineering
  University of Maryland, Baltimore County
  Baltimore, Maryland
- Ben Hosp
  Department of Computer Science
  George Washington University
  Washington, D.C.
- Jeremy Robin
  Cloudmark
  San Francisco, California
- Alan T. Sherman
  National Center for the Study of Elections
  Center for Information Security and Insurance

Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, Maryland

Lastly, we would like to thank uOttawa's GSAÉD and particularly chief retuning officer Angelika Welte for all of her support.

Bibliography

[1] Voting system performance guidelines. *2005 Voluntary Voting System Guidelines*, Volume 1, United States Election Assistance Commission, Version 1.0, 2005.

[2] Brennan Center Task Force on Voting System Security (Lawrence Norden, Chair). The Machinery of Democracy: Protecting Elections in an Electronic World. *Brennan Center For Justice*, 2006. `http://www.brennancenter.org/dynamic/subpages/download_file_39288.pdf`

[3] J. Buechler, T. Earnet, and B. Smith. Voting System Usability: Optical Scan, Zoomable, Punchscan. *UMBC CMSC 691/491V – Electronic Voting by Alan T. Sherman*, May 2007.

[4] D. Chaum. Secret-Ballot Receipts: True Voter-Verifiable Elections. *IEEE Security and Privacy*, IEEE Computer Society, 02.1, Los Alamitos, CA, USA, 2004, 38–47.

[5] D. Chaum, C. Crepeau, and I. Damgard. Multiparty unconditionally secure protocols. Proceedings of the twentieth annual *ACM Symposium on Theory of Computing*, 1988.

[6] J. Clark, A. Essex, and C. Adams. On the security of ballot receipts in E2E voting systems. Proceedings of *Workshop on Trustworthy Elections 2007*.

[7] J. Clark, A. Essex, and C. Adams. Secure and observable auditing of electronic voting systems using stock indices. *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) 2007*.

[8] A. Essex, J. Clark, R. Carback, and S. Popoveniuc. Punchscan in practice: an E2E election case study. Proceedings of *Workshop on Trustworthy Elections 2007*.

[9] K. Fisher, R. Carback and A.T. Sherman. Punchscan: introduction and system definition of a high-integrity election system. Proceedings of *Workshop on Trustworthy Elections 2006*.

[10] D.W. Jones. Chain voting. *Workshop on Developing an Analysis of Threats to Voting Systems*, National Institute of Standards and Technology, 2005.

[11] A. Kent. Unconditionally secure bit commitment. *Physical review letters*, American Physical Society, 83.7, 1999.

[12] Andrew W. Lo and A. Craig MacKinlay. *A Non-Random Walk down Wall Street*. Princeton University Press, 1999.

[13] Burton Malkiel. *A Random Walk Down Wall Street*. W. W. Norton & Company, 1973.

[14] Benoit B. Mandelbrot, Richard L. Hudson. *The (mis) Behaviour of Markets: A Fractal View of Risk, Ruin and Reward*. Basic Books, New York, 2004.

[15] A. Menezes, P. van Oorschot, S. Vanstone. Handbook of applied cryptography. CRC Press, Boca Raton, 1997.

[16] S. Popoveniuc and B. Hosp. An introduction to Punchscan. Proceedings of *Workshop on Trustworthy Elections 2006*.

[17] S Popoveniuc and J. Stanton. Undervote and Patter Voting: Vulnerability and a mitigation technique. Proceedings of *Workshop on Trustworthy Elections 2007*.

[18] S. Reiss. The Wired 40. *Wired*, 14.07, July 2006.

[19] J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63:9, 1975.

[20] RFR-002: VoComp Evaluation Criteria. Voting System Performance Rating Organization. July, 2007. Available online: http://www.vspr.org/rfr-docs/vocompe.pdf