Enabling More Meaningful Post-Election Investigations

by

Arel Lee Cordero

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy
in

Electrical Engineeing and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Fall 2010

Enabling More Meaningful Post-Election Investigations

Abstract

Enabling More Meaningful Post-Election Investigations

by

Arel Lee Cordero

Doctor of Philosophy in Electrical Engineeing and Computer Sciences

University of California, Berkeley

Professor David Wagner, Chair

Post-election audits and investigations can produce more transparent, trustworthy, and secure elections. However, such investigations are limited in cases by inadequate tools and methods, an absence of meaningful evidence, and high costs. In this dissertation, I address these concerns in the following three lines of research. First, I describe my research on verifiable and transparent random sample selection for post-election audits. I investigate how counties have typically approached random sample selection, and I analyze the implications and limitations of those approaches. I propose a sampling method that has since found use in counties across the country. Second, I describe a novel approach for logging events in direct recording electronic (DRE) voting systems. My approach gives investigators more meaningful evidence about the behavior of DREs on election day. In particular, I propose to record interactions between the voter and the voting machine such that they can be replayed by investigators while preserving the anonymity of the voter. Last, I describe a novel process for efficiently verifying elections that use optical scan voting systems. My process uses image superposition to let an investigator visualize the content of many ballot images simultaneously while allowing individual treatment of anomalous ballots. I evaluate this process and demonstrate an order of magnitude improvement in the time it takes to inspect ballot images individually. This approach will let investigators more cost-effectively verify that all ballots have been accurately counted as intended by the voters.

For my parents, Carmen and Rogelio

who ingrained in me the sense of equality that democracy aspires to

Contents

List of Figures

Acknowledgements

This dissertation was only possible with the support of many people. My advisor, David Wagner, is foremost among them. His incisive observations, guidance, and mentoring shaped not only my thesis, but also my perspectives on research and life. The professional and personal values he exemplifies I strive to emulate, and I would feel lucky to ever come close. I am tremendously thankful for David and for my other committee members, Deirdre Mulligan, Jitendra Malik, and Doug Tygar, whose work and ideas shape my research.

I attribute my interest in voting largely to the 2004 General Election and to my dissatisfaction with its transparency, if not its outcome. Also its outcome. That I realized I could do something about it I attribute to David Dill, who deflated (by way of an excellent talk) my original hope of finding a technological silver bullet for electronic voting, and then inspired me (along with David Wagner's graduate security course) to think of ways to improve the situation. Many people influenced my perspective on voting, but I especially acknowledge Matt Blaze, Joe Hall, Aaron Burstein, Ka Ping Yee, Cynthia Sturton, Naveen Sastry, Chris Karlof, Eric Rescorla, Dan Wallach, Elaine Ginnold, Peter Neumann, and Doug Jones. Ping, Cynthia, Naveen, Chris, and my other contemporaries in David's group, Erika Chin, Matt Finifter, Adrienne Felt, Adrian Mettler, Wil Robertson, Karl Chen, and David Molnar, all left a lasting impact, and I wish to acknowledge them.

I particularly wish to acknowledge my co-authors of this research: David Dill, Keaton Mowery, Alan Tsai, Theron Ji, and David Wagner. I am thankful for the opportunity to have mentored Keaton, Alan, and Theron, as well as my current and equally outstanding students, Eric Kim and Raji Srikantan. We would like to express our deep gratitude for the Alameda, Marin, Humboldt, and Yolo County Registrar of Voters offices for the generous time and effort they have provided to help us better understand the process of voting in California. Steve Bellovin and Gary Tyson first suggested the idea of replayable logs to us, and we gratefully acknowledge their contributions. We thank Juan Gilbert and Prabal Dutta, for their help and contributions on that paper as well.

My graduate career, and this research, were financially supported by generous people at Lucent Technologies, with the Bell Labs Graduate Research Fellowship; Tom Siebel and the Siebel Foundation, with the Siebel Scholars program; the University of California, Berkeley, with its Graduate Fellowship; as well as the National Science Foundation and its ACCURATE center. I gratefully acknowledge their support and the opportunities their support gave me.

Navigating the Ph.D. program was overwhelming at times—most of the time, in fact. I could not have succeeded without the encouragement and training of many people. Jan Cuny, Eugene Luks, John Conery, and Kent Stevens mentored me through my undergraduate computer science career at the University of Oregon, and inspired me to continue on to graduate school. Warren Dixon introduced me to research at Oak Ridge National Lab. Paul Adams and Roland Freund mentored me in research as well. At Berkeley, Sheila Humphreys, Juan Meza, David Wessel, and Jerry Feldman provided me with invaluable guidance and mentoring. La Shana Porlaris was instrumental to making me feel at home in the CS department.

I would be nothing without the love and support of my wonderful family and friends. I owe everything to the dedication of my mother and father to our education, and their innumerable sacrifices they made for my siblings and me. My family means the world. I especially wish to acknowledge my parents, Rogelio and Carmen, my brother Ro-el, my sister Nakai, Patrick, Consuelo, the D.J.'s, the Ric's, and my grandparents Betty and Roque—who set the standard. On my mother's side, I wish to recognize my aunt Irasema and her family, my uncle Leo—who personified art, my uncle Walter and his family, Raul Varela and his family, Luz Colmenares Martz, and my grandparents Carmen Varela and Leo Colmenares Varela.

I wish to acknowledge people who advised me, cheered me on, sacrificed for and with me, and inspired me through various parts of the tumult of a graduate degree. Though there are too many to list all, I wish to acknowledge Parvez Ahammad, Devdatta Akhawe, Norm Aleks, Oscar Arana, Alex Birch, Tamara Blain, Peter and Zuzka Bodik, Donald Chai, Charu Chaudry, Karl Chen, Jack Culpepper, Shereen Daly, Alex Fabrikant, Mayra Garay, Lucy Goodnough, Rachel Gross, Kat Hall, Teresa Hernandez, Ji Yun Jeong, Sana Kenshil, Psyche Loui, Adrian Mettler, Joe Milton, Goh Nakamura, Stephen Nelson, Shruti Shah Mody, Jason and Cleo Naranjo, Blaine Nelson, Pam Ng, Glen and Elaine Olsen, Matt and Julie Piotrowski, Mariana Raykova, Azucena Rodriguez, Rusty and Dara Sears, Gabriela Serrano, Mao Taketani, Aarti Tanna, Yumi Thomas, Michael Toomin, Colin Turner, Mario Vargas, Joel Weinberger, Lisa Westmoreland, Jeff Whiting, Jose Carlos Zavala.

Last, I wish to acknowledge everyone I forgot to mention, and I—with my sincerest apologies—offer a doubly-grateful acknowledgement to each affected person.

Chapter 1

Introduction

Democracy demands free and fair elections. Freedom and fairness are also not enough: elections must *persuade* enormously powerful stakeholders of their impartiality. They must operate in the face of extraordinarily high stakes and commensurate efforts to sway their outcome. For voters and candidates to accept the legitimacy of the vote, and accept any unfavorable consequences it may hold for them, election systems and processes must be sufficiently transparent, trustworthy, and secure.

In this work I explore ways of improving the transparency, trustworthiness, and security of elections—particularly for electronic voting systems in use today—by focusing on what happens *after* an election. I focus on three lines of research that enable better after-the-fact analysis of elections—better *post-election investigations*. My goal is to enable more meaningful post-election investigations that can detect, confirm, deter, and in some cases correct many of the things that can go wrong in an election.

1.1 Elections: a unique security problem

When I tell people I work on the security of voting systems I am sometimes asked what is so hard about the problem. I hear sentiments like, “why not give voters receipts like at ATMs?” Or, “God forbid we let people vote by Internet!” After all, while many security concerns exist with both banking technology and Internet commerce, enough checks and balances keep the systems working well enough. When unfortunate events do occur—when accounts are hacked into or credentials are stolen—we have mechanisms for mitigating those problems. These opinions certainly have merit as they represent those of perfectly legitimate stakeholders in our elections, but they miss a point that may not be obvious at first: elections are different.

Anonymity in elections. Elections pose a unique security problem for several reasons. One reason is the role of anonymity in elections, particularly in countries like the U.S. that use a secret ballot. The secrecy of the ballot, which frees voters to vote without coercion or fear of personal retaliation, sets up a tension between secrecy and transparency. Each voter, knowing only how they individually voted, must trust that the remaining votes were cast as intended and counted as cast.

In the U.S. the secret ballot is not only an option, but a requirement: voters should not be allowed to prove how they voted. This is to prevent the voter from selling his or her vote, and to protect the voter from being pressured or coerced to demonstrate how they voted. This requirement limits the use of common techniques for accountability like giving voters receipts showing how they voted. While such receipts could make it easier to audit an election's correctness, their use could undermine the impartiality—and thus the purpose—of the election.

Voter anonymity also has implications for accountability should something go wrong. The secret ballot, coupled with technologies or processes that are not fully transparent or trustworthy, can result in an absence of hard evidence to prove whether an election was corrupted or by how much. Even if allegations of corruption cannot be proven, the resulting public doubt can undermine the legitimacy and authority of the elected parties.

Conflicts of interests. Elections are also unique in the enormity of the stakes that hang in their balance, both locally and globally. Elections serve as a check and balance to the incumbent power, as leverage against a government inimical to the majority's interest. The most important elections have few—if any—neutral parties with nothing to gain or lose from the outcome. An election system must therefore tolerate the natural conflicts of interests that arise from stakeholders in an election running the election. For example, the Chief Elections Officer of Ohio in 2004, the Secretary of State of the deciding state in that year's presidential race, was also an active honorary co-chair of the George W. Bush campaign, raising accusations of an unfair bias. For instance, his strict interpretation of provisional ballot laws was argued in a lawsuit to disproportionately disenfranchise urban voters who trended toward John Kerry (4). Importantly, such conflicts of interest, if not always as conspicuous, are unavoidable. A free and fair election must be resilient to situations in which vested interests run the election.

Conflicts of interest also run through the makers of voting systems. Election systems vendors not only produce the technology we vote on, but in many cases are directly involved in running elections as well (77). Although the systems they produce should be neutral, the people who run these companies may have political inclinations, preferences or interests. For example, in the same election, the CEO of Diebold—one of four major vendors of election systems at the time—was quoted as being "committed to helping Ohio deliver its electoral votes to the president" (41). While it is rare for anyone to be as candid about a conflict of interest, we cannot depend on vendors to be immune to political bias. Further, conflicts at the executive level are not the only ones that are concerning. Employees designing and implementing the systems, or others with insider access, are also trusted with respect to the integrity of elections. For example, a backdoor inserted by one programmer, or an undocumented feature exploited by one worker, could be sufficient to compromise the vote on a large scale. Even if such conflicts of interest were prohibited by U.S. law, which they presently are not, accusations of bias would be easy to raise and difficult to rebut. For these reasons, voting systems must be transparent and robust against politically biased vendors.

Transparency and technology. Elections must be transparent, as transparency is what lets their many diverse and competing stakeholders come to trust their outcomes. In his dissertation, Joe Hall defines transparency in terms of four elements.

> "A fully transparent election system is one that supports *accountability* as well as *public oversight*, *comprehension* and *access* to the entire process." (37)

This is an ambitious goal; we must still work toward fully transparent elections. Some current practices, though, take us partway. Paper ballots, where used, are relatively comprehensible (both in terms how they operate, and what their limitations are). Mandatory random audits provide a degree of public oversight. Language diversity of ballots provide accessibility for non-English speakers. Assistive technologies empower disabled people to vote unassisted.

However, new technologies—electronic voting machines in particular (see Section 1.2)—have come at a price of transparency. Direct recording electronic (DRE) voting machines, for instance, leave no physical voter-verified record of the vote. People must therefore trust the machine to operate correctly: voters must trust that their votes were cast as intended, and all the parties must trust that the votes were counted as cast.

One factor affecting the transparency of new technologies is complexity: as systems get more complex they become increasingly hard to reason about, especially in terms of security and reliability. Most electronic voting systems used in the U.S. are enormously complex, especially compared to their relatively simple functional requirements. For example, the use of commodity operating systems like Windows or Linux in some voting machines expose those machines to a much larger attack surface: bugs in the operating system become bugs in the voting machine.

Complexity is only one problem affecting election systems in use today. Security reviews of widely-used voting systems have shown a pervasive and dangerous reliance on security through obscurity. Upon inspection, these voting systems have revealed abundant design flaws ranging from the misuse of cryptography, to ineffectual physical security (52; 17; 57; 36). Researchers have repeatedly demonstrated attacks on these systems that could arbitrarily change election results, and be difficult if not impossible to detect (44; 17; 19).

Also compounding the problem is an issue of trust: electronic voting systems in use today require placing a lot of trust on many people. When vote records or voting machine software can be electronically altered, one alteration by one rogue programmer could swing an election (44; 17). This dispersion of trust, this escalation in the number of potential points of failure, raises the threat of large-scale compromises of an election.

Elections should be transparent so all stakeholders can convince themselves of its fairness. The evolving use and misuse of technology in elections—particularly with respect to electronic voting systems—has hurt transparency. In this thesis I have favored lines of research that apply to today's voting systems, and that improve their transparency and security with only incremental changes to the existing infrastructure and policies. While approaches that involve more extensive technological and procedural changes are certainly promising[1], new voting paradigms face steep obstacles to adoption (see Section 1.2.1), thus increasing the importance and urgency of improving the systems in use today.

1.2 Voting machines in use today

The U.S. uses many kinds of voting machines, as the federal government does not mandate one particular method or system of voting. The industry is composed of many private companies, though only several dominate. Four out of every five jurisdictions in the U.S. use systems by one of the four largest vendors: ES&S, Dominion (formerly Premier/Diebold), Sequoia, and Hart InterCivic (79). Of these systems, two classes of systems are most popular. These are detailed below.

Direct recording electronic systems. Proposals for *direct recording electronic* (DRE) voting machines have existed since the mid-19th century, although their first use in mainstream elections dates to the mid-1970s (49). What sets DREs apart from other voting systems is how the official record of the vote is stored. Instead of a physical ballot, records are stored electronically. In modern DREs this usually means on a reusable (and often removable) memory card.

DREs are not the first voting systems to separate the record of the vote from the physical medium the voter sees. For example, *lever machines*—mechanical devices that also store vote counts internally—are still used in New York where they were introduced at the end of the 19th century (49).

The advantages of DREs are numerous (at least in theory). For example, votes are generally unambiguous, easily tallied, re-tallied, and reported. No handling of physical paper is required. Electronic voting machines also allow for a more flexible user-interface. Modern DREs, such as the Diebold AccuVote-TSX touchscreen system, are built on general-purpose operating systems (the AccuVote-TSX is built on Windows CE) (80). This flexibility makes it easier to support visually- or motor-impaired voters, as well as multiple languages. DREs also offer the promise of fewer consumables, a justification cited for amortizing their high price tag (61), although the hoped-for cost advantages have not always panned out (11).

DREs have been the subject of increasing detraction, particularly with regard to their

[1]For example, end-to-end voter-verifiable voting systems offer the promise of letting each voter verify that his or her vote was counted, and counted anonymously (18; 68; 66; 8).

record of poor security and reliability, and nearly complete lack of transparency inherent to the unverifiable way vote records are stored on today's DREs (29; 17). The complexity of modern DREs, the modifiability of the voting machine and election-management software, and the implicit and explicit communications channels used in managing DREs, create avenues for systemic failures, and wholesale attacks on the election machinery. Further, because of lack of transparency, failures can go unnoticed, and leave behind little, if any, evidence of a problem (29; 17; 24).

Attempts to improve the transparency of DREs have gained traction, most notably by requiring a voter-verified paper audit trail (VVPAT). However, 17 states still use DRE systems without a VVPAT (79). Research on the use of VVPAT subsystems also raises doubts on their efficacy (28).

Optical scan systems. Optical scan ballots are paper ballots designed to be marked by hand and counted by machine[2]. They account for about two-thirds of votes cast in the U.S. today (79). The main advantage of optical scan systems over hand-counted paper ballots is the automation the scanners enable. Instead of people manually counting each ballot, the tedious work can be offloaded to machines (either at the time of casting, as in *precinct-count* optical scan systems, or after all the ballots have been aggregated, as in *central-count* systems).

While machines are capable of tallying ballots with high speed and accuracy, using machines introduces a problem of trust. Since the machines rely on software that must be configured and calibrated for each election, errors can accidentally (or deliberately) occur that can affect the tallying or reporting of a large number of ballots.

However, unlike votes stored electronically or magnetically on a DRE, optical scan ballots are physical records of the voter's intent that can in principle be reexamined in an audit, or recounted. This does not mean paper ballots are immune from failures. Paper ballots must be securely managed, handled, transported, and preserved. This requires stringent procedural controls and regulations.

The weaknesses of today's voting machines undermine the fairness and persuasiveness of our elections, requiring us to either improve the machines we have, or secure the integrity of the election despite our flawed machines.

1.2.1 Obstacles to improving voting systems

Many obstacles stand in the way of improving the security and reliability of today's voting systems. Because there is no single standardized method or system of voting in the U.S. (each state or county decides what to use), addressing the voting problem involves considering many systems, and problems that arise in each.

[2]*Ballot marking devices*, such as the ES&S AutoMARK, are related to optical scan systems in that the voter votes on a DRE-like device (e.g., a touchscreen), and the device produces a marked paper ballot that the voter can then cast as an optical scan ballot.

The proprietary nature of commercial voting systems also has drawbacks, particularly for researchers in the field. For example, vendors have generally not made the source code for their voting machines available to academic researchers for research purposes. It took an accidental leak of source code to enable the first published security analysis of a modern touchscreen voting machine (39; 52). It took the mandate of California's newly elected Secretary of State to enable a further security study of voting systems used in the state (17).

Another obstacle is certification. While voting systems in the U.S. are purchased at a local level, all systems must pass state and federal certification. The high cost of certification, however, creates a high barrier to entry for new voting system vendors, and makes it harder for existing vendors to improve their current systems (7). Certification also adds a high cost and delay to updating systems, and patching security vulnerabilities in a timely manner.

Policy and election laws, which vary state by state, also complicate matters. Transparency improvements viable in one state may not apply to another without a substantial legal and political effort[3].

Researchers must take into consideration these obstacles, as well as the interdisciplinary nature of the field. These obstacles inform the lines of research I take, in particular by favoring results that require few technological or procedural changes to existing systems, but benefit the transparency and security of elections.

1.3 Post-election investigations

Election incidents cannot always be prevented, and in the same way airplanes carry "black boxes" to record forensic evidence in the unlikely event of a crash, election systems must also preserve evidence in the event of a failure. However, unlike an airplane crash, even catastrophic election failures with enormous geo-political consequences may be difficult to detect or confirm.[4] For this reason, regular investigations or audits of elections are necessary, and these require trustworthy evidence.

The goals of post-election investigations are:

- **Detect** possible election failures (e.g., miscounts of the vote).
- **Correct** any inaccurate election outcomes before certification, if possible.
- **Confirm** the outcome of an election in a way that allays all doubts about the legitimacy of the vote. (Even if no election incident occurs—doubt from any of the stakeholders should be allayed through evidence and investigation, for instance, a manual recount.)

[3]For example, Virginia has very restrictive rules that make post-election audits "nearly impossible." (72)

[4]Indeed, some electronic voting machines in use today do not even make it possible to confirm or refute the incidence of a machine failure (85).

- **Deter** election manipulation—dissuade people from committing fraud—by making it improbable to get away with it.

To accomplish these goals, elections must record trustworthy evidence of the voters' intent, and the investigations themselves must also be transparent and trustworthy. Post-election investigations must be inexpensive and efficient enough to be performed regularly, and comprehensive enough to be effective. Investigations may come as formal requirements in the election code, or as informal responses by officials or other parties who may be suspicious of an anomalous or unexpected result. One common type of post-election investigation is the mandatory random audit.

1.3.1 Random audits

Post-election random audits (also known as manual tallies) are instrumental towards more verifiable, trustworthy and secure elections. Random audits are conducted after the election, but before its certification. As such, they can be used to detect problems in time to correct them for the final result. In many states, random audits are a regular procedure after elections: as of writing, 25 states require audits (79).

An audit works by focusing attention on a randomly selected sample of precincts or batches of ballots (precincts often contain on the order of $1,000$ ballots). The selected precincts are inspected carefully, and recounted. By the time of the audit, the county should have already tallied each precinct once, and the audit seeks to confirm the original count.

If the original tally was incorrect—for instance, if the voting machines miscounted a large number of ballots—an audit of the affected ballots would have some probability of finding a discrepancy. If the audit detects a discrepancy that cannot be explained or reconciled, the audit could trigger a further investigation or escalate to a full manual recount.

The number of precincts audited is usually a small subset of the total number. The exact number depends on the state. For instance, the number might be fixed by legislation (e.g., 1%, 3%, etc.), or it might depend on circumstances such as the margin of the race (e.g., close races may demand larger audits).

For an audit to detect *intentional* manipulation of the vote totals (i.e., fraud), the audit must assume that an adversary will choose a strategy that minimizes her probability of detection. This introduces several subtleties to the audit process. First, it is important that an adversary be prevented from changing the original tallies *after the fact* to make the audit check out. Second, the selection of ballots in the audit must be random *and unpredictable* to any adversary.

The purpose of the audit is to increase confidence in the election outcome and to deter fraud. This can be formulated as a statistical problem such that the result of auditing X ballots results in Y degree of confidence that a particular election was decided correctly. This is the approach that risk-limiting election audits take (47).

An audit or recount relies on having a verified record of the vote that serves as the ground truth. A DRE without a voter-verified paper audit trail, for example, does not have such a record. Therefore, there is a limit to what an audit could accomplish in this case. However, if meaningful, auditable evidence exists, an audit can provide a robust tool for verifying an election's integrity.

1.3.2 Logging subsystems

Event logs are commonly used by investigators when investigating electronic voting systems. Voting systems record important events in logs—*audit logs*—that can later serve as evidence should the voting machine be suspected of failing, or as part of a routine audit. Events such as rebooting the voting machine, starting an election, casting a vote, or closing an election are usually recorded on some kind of persistent storage, along with the time and date of occurrence. If a machine stops working in the middle of an election, for example, the audit logs might give investigators some information as to the prior state of the machine.

Logging subsystems in machines today, however, have two large deficiencies: they suffer from poor design and engineering choices that limit their reliability; and the events they do record are not very expressive, reducing their usefulness in diagnosing or detecting many failure modes in post-election investigations.

For example, voting machines today store logs on rewritable media that can be reset, erased, and modified (17). Cryptographic techniques that could address the authenticity and integrity of the media, if used, are often used inappropriately (17). This makes it possible for a malicious person to change the record of events, and avoid detection. Even in the non-malicious case, software bugs or human error can unintentionally corrupt or erase log records (14).

Even excusing the issues of integrity and authenticity, logs do not always record the right kind of information. For example, the investigation following the "deck-0" flaw in Humboldt (see Section 1.4.3) found event logs to be practically "useless" (88). Another study of the AV-OS voting system found its logs to contain only time stamps (not time *and date* stamps) for most events. This creates ambiguities if a voting machine is powered on for more than one day, as is common (9).

Another engineering problem with today's logging subsystems is the lack of separation between the logging subsystem and the rest of the voting machine software (17). If the voting machine software gets corrupted, the logs can also get corrupted. A software error, or a malicious attack—events for which event logs would be especially useful—may result in logs that are inconsistent with what actually occurred. When this is possible, the trustworthiness of such logs is diminished.

Logging subsystems today are not reliable to the level desired or expected of such a critical system. If logging systems are improved, however, they could be very useful for increasing the integrity of elections.

1.4 Motivating election incidents

Recent election history is littered with illustrative examples of how elections fail. Some failures are silent and are never discovered, for instance if no audit exists to detect one. Other failures are relegated to speculation because the evidence needed to confirm a failure may be irrecoverable or insufficient, as could easily happen with DREs that record evidence on rewritable and unauthenticated media. The best failures, however, we can learn from. They become cases in point that provide a clear problem statement for future research. In this section, I briefly describe three such cases that motivate my research and this thesis.

1.4.1 Cuyahoga County, Ohio 2004

The first example motivates my work on transparent sample selection in post-election audits (see Chapter 2). The setting is the 2004 General Election in Cuyahoga County, Ohio—the deciding state in the narrowly-contested race for President (23). In 2004, Cuyahoga was still a punch-card county (83). Like optical scan—or other machine-tallied paper ballot systems—punch-card systems are subject to systemic failures, for instance if the machines counting them are misconfigured. A post-election audit (see Section 1.3.1) can detect such failures with some probability by manually recounting a subset of ballots and checking that the machine tallies match the manual tallies. From the results of an audit, one can make inferences about how the remaining ballots were counted. If the audit detects a problem—if the tallies cannot be reconciled—further investigation is warranted.

In Cuyahoga, no problems were directly detected by the audit, but this in fact became the source of controversy. Instead of selecting the random sample of ballots in the presence of observers, election officers quietly selected the random sample in advance, when no observers were present. Then, the election officers secretly recounted all the selected ballots to ensure that no problems would be discovered during the public audit. When observers finally arrived for the official event, by design no further discrepancies were detected (31).

In so doing, the officials subverted the transparency goals of the audit, ensuring that the public would remain unaware of any problems that might have been found by a properly conducted audit. The officials' secret actions were detected only when an observer noticed that the audited ballots already appeared to be sorted by candidate (an unlikely occurrence by chance). An investigation subsequently found the officers guilty of felony offenses for deliberately subverting the audit (31).

A takeaway lesson from this example is that audits must be random *and unpredictable* to any possible adversary—including election officials. Though many election officials certainly are trustworthy, they should not have to be blindly trusted: public observers need a way to verify the randomness and unpredictability of the audit, without blind reliance on election officials. This poses requirements on the method of selecting ballots, because the efficacy of the audit hinges on an honest random selection.

1.4.2 Sarasota County, Florida 2006

The second example motivates my work on replayable log systems for electronic voting machines (see Chapter 3). The case involves the 2006 race for Florida's Congressional District 13 (CD-13). The race, which spanned five counties, came down to a razor-thin margin of 369 votes between the two leading candidates (24).

The largest county, Sarasota, which accounted for about half the votes of this race, experienced an anomaly. An unusually high number of voters—roughly 15% or $18,000$ voters—did not cast a vote in the CD-13 race, compared to the 2.5% or $3,000$ undervotes observed in the remaining counties. This abnormal undervote rate was not seen in any other comparable contest or in votes cast on paper, and many voters complained that the voting machine never gave them a chance to vote in the CD-13 contest. Troublingly, the demographics of Sarasota county leaned toward the favor of the losing candidate. A statistical analysis found that the high undervote rate likely changed the outcome of the race (22).

Complicating the problem was the use of DREs in Sarasota County. The ES&S iVotronic devices used in 2006 were problematic in several ways. First, these machines did not have a voter verified paper audit trail, limiting the evidence investigators had to understand the anomaly. The system event logs (or audit logs), which could have yielded a more conclusive investigation of the CD-13 anomalies, did not record sufficiently meaningful information. This forced investigators to rely more on other less-direct approaches, such as software inspection, which is a hard problem (85).

A takeaway from this example is that voting machine audit logs, if better engineered, could be much more useful than they are in today's systems. If the logs do not record all relevant events, or cannot be trusted to be accurate, it becomes much more difficult to discover and explain problems when they happen. Logs that did accurately record all relevant events would help enable more definitive post-election investigations.

1.4.3 Humboldt County, California 2008

The last example I describe motivates my work on developing an interactive image-analysis tool for optical scan paper ballots (see Chapter 4). Beginning with the June Primary Election in 2008, the Humboldt County Election Transparency Project (ETP) began independently rescanning and publishing images of each ballot cast (42). Using a commercial scanner, the county methodically made an image of every ballot, in addition to tabulating them as normal with the official optical scan system. By publishing the ballot images, any interested person could then inspect and recount the ballots, and compare their results to the official reports.

Elections, however, can be quite large. For example, the 2008 General Election in the relatively small county of Humboldt, California had $64,000$ ballots cast. Further, each ballot contains many contests, each of which must be separately tallied. (The same election had 36 contests.) This makes it unlikely that any one person could manually count all these ballots in a reasonable amount of time. To more efficiently recount ballots, the Election Transparency Project wrote their own software to independently recount the ballot images (78).

In the following 2008 General Election, the Election Transparency Project discovered a discrepancy in the vote totals. The independent recount found that 197 ballots were missing from the official results. The official election management software, written by Diebold/Premier, had mysteriously dropped the first deck of ballots (42).

This discovery led to an investigation of the Diebold/Premier system used by the county (14). The investigation found a pervasive flaw in a common version of software used in California and other states that in fact did drop the first deck of scanned ballots, when counting ballots centrally. As a result of this discovery, the missing votes were recovered, and the Secretary of State of California decertified the deficient software (16; 15).

This case shows how even paper-based systems can be prone to systemic failures when they rely on complex hardware and software for tabulation. It also shows the benefit of using independent software to validate an election by recounting ballot images. While investigators could use the images to recount the election themselves, independent software can speed up the process. This motivates our research on methods for auditing election tallies using these ballot images.

1.5 Contributions

My thesis work includes three lines of research.

1.5.1 Transparency in random selection for audits

Random audits of election systems can be used to statistically verify the integrity of an election by inspecting and recounting a portion of the cast ballots (see Section 1.3.1). Random audits approximate a complete manual recount, but are more efficient because only a subset of ballots are recounted. Random audits rely on the properties of the random selection to substantiate claims of an election's correctness, and many states require such audits as a standard practice (see Section 1.3).

For the results of a random audit to be meaningful, however, it is critical that the *random selection*—i.e., the choice of precincts or ballots to be recounted—be performed transparently and in a publicly understandable and verifiable manner. In my research, I discovered that this was often not the case, and that election officials lacked tools and procedures to meet these requirements.

Distinctions that might be well-understood by people familiar with the subtleties of randomness may not be obvious to all election officials, let alone the general public. For instance, the importance may not be immediately apparent of the distinction between selecting an *arbitrary* precinct to audit versus a uniformly *random* one. An arbitrary sample selected by, for example, a well-intentioned election official subverts an audit not only by introducing a bias that reduces its statistical power (as discussed below), but also by making it hard for observers to verify the selection's *unpredictability*. If an adversary can predict which precincts will be selected, he can fully subvert the remaining ones without detection by the audit. In practice, election officials have commonly confounded randomness with arbitrariness, often imposing strong biases for smaller precincts[5], or for precincts that would avoid detecting known problems with the tallies (31).

My research also found that election officials often turned to computer programs to select the random samples. However, I showed that this was a bad idea not necessarily because of the common use of insecure pseudo-random number generators (such as linear congruential generators), but because the opacity of *any* algorithm performing the selection makes it hard to assert to the public that the sample was unpredictable.

In Chapter 2 I analyze methods for random selection for auditing, including several methods used in practice. I show that the methods in common use at the time of this work fail to provide transparency or public verifiability. To address this shortcoming, I propose a simple, publicly verifiable random selection procedure. This scheme balances the need for a verifiably uniform distribution with the requirement of being easy to comprehend by the public, by using dice and tables. Since this research was completed, this method has been adopted in several counties in California and elsewhere, and has been favorably recommended in several subsequent studies of transparent election auditing (38).

1.5.2 Replayable audit logs for DREs

Post-election investigations are only as meaningful as the evidence available to investigators. Electronic voting machines—DREs, in particular—have a poor record of recording meaningful evidence. For example, DREs with no voter-verified paper audit trail (VVPAT) require investigators to heavily trust the system event logs. However, these event logs, the software that records them, and even the set of events recorded, have been shown to be unreliable (17; 9; 14). In this research I look at an approach for improving logging in voting machines.

In Chapter 3 I describe a new approach for logging voter interaction with DRE voting machines. My approach, which I argue records a more complete and meaningful view of the election-day behavior of voting machines, is based on recording the I/O of a DRE to capture the DRE's behavior as seen by the voter. The event log is designed so that post-election investigators, with minimal effort, can later *replay* all recorded events, and compare what the voting machine recorded with what the voter saw.

[5]In my personal interaction with election officials, they told me that often when they were asked to randomly select several precincts, they would tend to pick the smallest ones, to reduce the amount of work for their staff.

One benefit of our approach is that, because our logging subsystem is smaller and more specialized than a voting machine (even a streamlined one like Pvote (86) on which this work is based), there is less code in it to trust. This makes the code easier to inspect, and has the effect of reducing the trusted computing base (TCB) of the voting machine. In particular, that our logs can be used to detect a misbehaving voting machine that interacts with a voter one way, and records her vote another way.

An issue that arises in this work is a tension between collecting better evidence for post-election investigations, and preserving the voter's anonymity. I investigate this tradeoff in an implementation, and demonstrate an approach that balances both requirements.

1.5.3 Efficient User-Guided Ballot Image Verification

The third part of this thesis work, presented in Chapter 4, deals with optical scan paper ballot systems, the most prevalent and increasingly popular method of voting in the U.S. (79). Optical scan systems have several desirable properties over alternatives in use today. First, voters vote on paper ballots, which preserve an auditable and physical trail of evidence of the voters' intent. Additionally, optical scanners automate the vote tallying process, replacing the otherwise tedious and labor-intensive process of hand counting. Despite their benefits, however, optical scan systems are vulnerable to failures that can result in miscounted votes and lost confidence.

One class of failures stems from misconfigured software. A misconfigured scanner, for instance, may have an inaccurate model of how votes are represented on a ballot, or which ovals correspond to which candidates, leading to lost or misattributed votes. Because of the complexity of software in modern systems, subtle and inadvertent errors may occur that do not emerge during pre-election logic and accuracy testing. For example, a configuration error in the 2006 Primary Election of Pottawattamie County led to a vast number of votes for one candidate being unintentionally assigned to another (67). Configuration failures can be difficult to detect automatically because re-tallying the ballots based on the same incorrect configuration information may lead to the same incorrect results.

One way to detect optical scan failures is to recount the ballots by hand. This often happens as a matter of routine in the form of random audits. However, routine recounts of *all* the ballots are rare because of the expense both in time and money (33). Thus, in practice, only a small subset of ballots get the individual attention necessary to detect many possible failures.

Because manually re-counting every ballot is expensive, an alternative that has been suggested is to rescan all the ballots with an independent commercial scanner. Such a scanner need only record images for each ballot. With images in hand, independently written software could tally the ballot images and provide an independent check on the correctness of the optical scan system (42). While valuable, however, independent software is still vulnerable to many of the failure modes that affect the original system, and does not do away with the need to configure the software.

Alternatively, the ballot images could be recounted by a human inspector. However, while electronic images may more manageable than paper, inspecting a large number of images is still tedious and slow. In Chapter 4 I present an approach to *efficiently* inspect optical scan ballot images roughly an order of magnitude more quickly than before. The tools and techniques I present let a person visually recount a contest from superimposed ballot images, simultaneously inspecting many ballots at a time. My approach also allows anomalous or ambiguous ballots to be easily identified and dealt with individually. This provides a powerful mechanism for after-the-fact analysis of elections conducted using optical scan voting systems.

Chapter 2

The Role of Dice in Election Audits

Random audits are a powerful technique for statistically verifying that an election was tabulated correctly. Audits are especially useful for checking the correctness of electronic voting machines when used in conjunction with a voter-verified paper audit trail (VVPAT). While laws in many states already require election audits, they generally do not address the procedure for generating the random sample (74). The sample generation procedure, however, is critical to the security of the audit, and current practices expose a security flaw. This chapter examines the problem of sample selection in the context of election audits, identifies necessary requirements for such a procedure, and proposes practical solutions that satisfy those requirements.

2.1 Introduction

Many things can go wrong in an election, whether intentionally or unintentionally. The ability to detect (and correct) errors is critical. Recent decades have seen widespread use of computers and automation in elections, including use of optical scan machines, DRE (direct recording electronic) machines, and computerized election management systems to record, tabulate, and report votes. Unfortunately, this trend has come at some cost to transparency, as the automation of these processes reduces opportunities for observers and interested members of the public to monitor the operation of the election. Random audits, performed after the election but before certification, remain as one of few defenses for ensuring fairness and for building public confidence in the result. Consequently, the details of these audits are of increasing importance to election integrity.

When done right, and in a transparent and publicly observable way, random audits can establish *objective* and *quantifiable* measures of election accuracy[1] However, without a transparent process, there is no reason to believe an audit will correctly represent the elec-

[1]The theory of statistical polling and statistical quality control provides a quantitative measure of con-

Largest Cities	Oakland, Fremont, Hayward, Berkeley
Population	1,507,500
Registered Voters	714,490
Total Voting Precincts	1,140
Absentee Precincts	240

Figure 2.1. Some basic facts about Alameda County, CA.

tion, which would defeat objectivity, rendering it meaningless as an assurance of fairness. Even if correct, a non-transparent audit would have trouble quelling skepticism and could thereby fail to provide confidence in an election.

All parts of an audit must be performed correctly—and transparently—for the the audit to mean anything. In particular, getting the random sample selection process right is critical. To start with, the selection process must ensure every vote has an equal (or minimal) probability of being selected. Moreover, like any fair lottery, a second requirement is that no party be able to bias or predict the selection in any way. An important implication is: *for an audit to give every party confidence in an election, every party must also have confidence in the fairness of the sample selection.* In other words, sample selection can easily become a weak link in the security of an election.

A case in point is the 2004 U.S. presidential contest in Cleveland, Ohio (56). Cleveland election workers, in an effort to prevent a complete recount, surreptitiously preselected an audit sample (3% of the total precincts) to ensure the ballots recounted by hand would match the initial machine counts. Then, with public observers present, the workers faked a random selection and the preselected sample was used, defeating the purpose of the audit. Because of this deliberate or negligent action, the true result of the race in Ohio, the deciding state in the presidential race, may never be known. To that extent, the integrity—or security—of the election was compromised.

2.2 Background

In the United States, elections are conducted at a local level (5). In California, for instance, specifications for equipment are set by Secretary of State and are implemented by counties according to California election code (1). As residents of Alameda County, California, we have observed the election and audit procedures used there, and we will use California and Alameda County as a running example of current practices.

fidence in the accuracy of election results, which can be calculated as a function of parameters such as the sample size (60).

2.2.1 California

California election code has included, for the past four decades, a requirement for a 1% manual tally (or audit) after every election:

> §336.5 "One percent manual tally" is the public process of manually tallying votes in 1 percent of the precincts, selected at random by the elections official, and in one precinct for each race not included in the randomly selected precincts. This procedure is conducted during the official canvass to verify the accuracy of the automated count (2).

The process is *public*, meaning any citizen is invited to observe every step of the audit. This requires a transparent process. The 1% manual audit is performed as part of the official canvass, as one of the last steps before the final election results are certified. Typically, after the sample is selected, election officials print out a report containing the electronic tallies for just the selected precincts, recount by hand (using three- or four-person recount boards) all the paper ballots cast in each selected polling place, and check to make sure that the manual count in each precinct matches the electronic count in that precinct.

The law requires every county to randomly select a minimum of 1% of precincts for the manual recount, but does not stipulate *how* it should be done. As it happens, Alameda County (Figure 2.1), and presumably many other counties, have turned to computers to perform the random selection.

2.3 Sample Selection

Computers are generally inappropriate for generating random samples in an election audit. Excellent software-based pseudorandom number generators (PRNGs) exist, so it might not be obvious why this use of computers in *election audits* is inappropriate. The inherent problem is transparency: running software is not observable. In fact, this is the exact problem faced by DREs, and the exact problem election audits, together with voter-verified paper audit trail (VVPAT) systems, intend to address. The use of a computer as a random source jeopardizes the integrity of the audit and election.

One devastating threat is that an insider might be able to tamper with the software used for random selection in a way that allows him or her to know in advance the outcome of the selection. An insider could then use this to cheat without—or with a lesser probability of—getting caught. For instance, an insider with advance knowledge of the selected precincts will be free to defraud all other precincts without fear of detection. Or, if the insider has already tampered with the votes in some precincts, the insider could modify the computer's PRNG to exclude the possibility of choosing those precincts. These attacks could be mounted in a way that is very difficult or impossible for observers to detect. Without a verifiably random sample, it is not clear that such an accusation could be defended against.

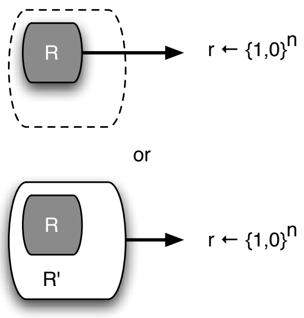


Figure 2.2. Transparency can be a problem regardless of the quality of the random number generator.

Dedicated hardware random number generators (perhaps based on physical phenomena such as radioactive decay, or radio static) are subject to the same transparency problem, despite typically being excellent random number generators. Without transparency, even with a perfect random source, it is hard or impossible to trust the authenticity of its output (Figure 2.2).

2.3.1 Requirements

As we saw above, "black-box" random sources are a vulnerability for election audits. What requirements must a viable solution meet? Many choices exist for generating random numbers; proving them unpredictable—in this case to *all* parties—is difficult, if even possible (71). What matters in the election setting, however, is choosing a procedure that can be used, understood, and trusted by an average member of the voting public, for instance, by an average high school graduate.

A transparent procedure for sample selection involves two problems:

1. *Transparently generating random bits,* and
2. *Transparently turning those bits into a sample.*

Technically, this may seem like a trivial distinction. However, the requirements that follow apply equally to both, and especially because the needs of a general audience must be considered, satisfying the requirements for both is not as easy as it might seem. Many natural schemes have non-obvious problems or pitfalls. In the remainder of the section we lay out several requirements for a transparent random source and selection procedure, and subsequently evaluate possible solutions against these requirements.

Simplicity. *The procedure must be simple to follow and execute.* The public must, without extensive education, understand the procedure, why it is fair, and why they should trust it. Otherwise, confidence in the audit and election may be limited. Additionally, complexity introduces opportunities for error or exploitation.

Verifiability. *The procedure must be verifiable either by inspection (i.e., it is physically observable) or by some other property (e.g., cryptography).* Verifiability is imperative. Every observing party must be assured that the selection is genuinely random with a satisfactory distribution. Otherwise, as in Ohio, the audit could be subverted, or observer's confidence undermined.

Robustness. *The distribution of the procedure must be hard to bias or manipulate.* It should be difficult for any party, particularly someone working from the inside, to affect or gain information about what set of ballots is included or excluded from the audit.

Efficiency. *The procedure must not take an incommensurate amount of resources to prepare or execute.* Election worker's and observer's time, energy, money, patience cannot be taken for granted. As the incident in Ohio demonstrated, the cost of time might tempt election workers to fudge the procedure. Or, perhaps, impatience could reduce an observer's vigilance.

2.4 Possible random sources

We first examine the applicability of several random sources to election audits, in light of the requirements above.

Cryptography. From the perspective of a cryptographer, the problem of verifiably-random numbers is (at least in principle) solved. One solution (51), for instance, is to have every observer pick a random number of their own and commit to it, perhaps by writing it down and dropping it in a box. When everyone has committed their numbers, they are revealed and summed modulo some number N. The result will be a random number if at least one observer was honest. This could then be built into a very reasonable scheme to perform sample selection.

To an extent, this solution meets all our criteria. However, if we consider trying to explain this to the general public and to election's officials, it might take some work. The concepts of modular arithmetic, independence, and uniformity of distributions must be understood. Also, cryptographic protocols often rely on participants to protect their own interests, which can be a problem when dealing with a non-cryptographer public. For instance, suppose instead of dropping numbers into a box, numbers are written on a board so that the attacker has the "last say," and thus deterministically chooses the outcome. The protocol assumes a savvy enough public to know that commitments are supposed to be hidden.

Drawings. A familiar method of random selection is a drawing in which items are mixed then drawn from a container. For instance, we could write the names of all the precincts on identical slips of paper, mix them well in a box, and draw our sample. This approach, however, makes verifiability difficult if many items are involved. For instance, if there are 1,000 slips for 1,000 precincts, election officials must convince observers that all 1,000 precincts are included. Any omissions would represent precincts an attacker could potentially subvert without consequence.

Drawings also have the disadvantage that it is hard to know when there has been sufficient mixing. For instance, the Vietnam draft lottery of 1970 used such a scheme, but was later discovered to have suffered from bias: birthdates later in the year were added last, and due to insufficient mixing, were more likely to be chosen earlier (75).

A much better approach is to perform the drawing with a small number of objects, such as ten cards or ping pong balls[2] and draw digits representing precincts (Section 2.5). This greatly improves verifiability and robustness, and is still simple and relatively efficient. Since only one digit is drawn at a time, however, the selection process can take an appreciable length of time for large elections.

Lottery-style drawings. Lotteries put a great deal of resources and creativity into maintaining secure random number generators. Lottery machines are culturally familiar and trusted to the extent that the lottery is played by millions of people. It would be prohibitively expensive, however, to replicate and maintain a lottery machine in each Registrar of Voters office, so if we were to create a sampling scheme from lottery numbers, we would want to use the same machinery—and perhaps drawings—used for the actual lottery.

Although simplicity, verifiability, and robustness are excellent, efficiency can become a problem and must be carefully considered in any scheme using lottery drawings. For instance, if a state lottery is chosen, this might impose a heavy travel burden on vigilant observers determined to witness the selection. Another issue might arise because U.S. elections are performed locally: if a single dedicated state-wide lottery were used for selection, coordination between county and state might be a problem (e.g., the lottery could only be done when the last county is ready).

Alternatively, the use of an agreed-upon future drawing as input to a deterministic algorithm that creates the sample, might make an excellent scheme. However, the algorithm must also satisfy the requirements in Section 2.3.1. Particularly, it must be verifiable and understandable to the general public.

Random number charts. Another idea might be to use a book filled with random numbers (64) as a basis for selection. Of course, printed books or charts are static documents and must be assumed to be known in advance to any attacker. Consequently, any methods that use books of random numbers would have to find another random method of selecting digits within the documents.

[2]Alameda County used this method quite successfully after the November 7th, 2006 General Election. The method was previously used for jury selection.

Cards. Cards are a time-tested source of randomness. Indeed, they are used in many high-stakes games. However, there are fifty-two cards in a deck, making it hard to verify—especially when subject to sleights of hand, such as those found in card tricks. As with drawings, it might be difficult to ensure the deck is sufficiently shuffled.

Another issue with cards is that not everyone is familiar with their properties (the suits, the ranks, the number of cards), nor how to handle or shuffle cards well. We would prefer a sample selection scheme that as many people as possible understand and could use.

Depending on how the cards are used, efficiency could also come into play, since every action needs to be observed and verified, and preceded by a shuffle.

Coins and dice. Coins are the quintessential random number generator and arguably the most ubiquitous. Methods even exist to mitigate any bias in a coin[3]. The primary drawback of coins is that their bandwidth is limited: generating many bits of randomness requires many coin tosses, which takes time.

Dice produce a higher bandwidth of random numbers[4], and have stood the test of time[5]. Dice are also available with different numbers of faces. Ten-sided dice are especially appealing because they map directly onto the decimal numbers[6]. Multiple dice can easily be rolled at once, significantly improving efficiency compared to single-digit drawings.

Dice, as with coins (or any random number generator above), can be biased—intentionally perhaps—so care must be taken to ensure fair dice (27). Techniques for mitigating potential attacks exist and include: using only new, translucent dice; using a ribbed tumbler to roll the dice; and rolling the dice onto a flat ridged surface such as a dice tray (Figure 2.3). Few objects are needed in a dice setup, however, which benefits verification.

In our proposed solutions that follow we use dice as the source of randomness because they best meet our requirements of *simplicity*, *verifiability*, *robustness*, and *efficiency*.

2.5 Creating the sample

Once a source of randomness is chosen, a practical procedure must be built around it to perform the sample selection. The procedure, like the random source, must satisfy our above requirements: it must be simple to use and understand, easy to verify, robust against tampering and reasonably efficient.

For the schemes we present, we opted for simplicity because that makes the other requirements easier to reason about. While designing a procedure for Alameda County, our

[3]Assuming independence of coin flips, *pairs* of flips from the same coin can be used to eliminate any bias the coin may have. If we throw out any HEADS-HEADS or TAILS-TAILS combination, we are left with two outcomes that bare equal probabilities. This idea is attributed to John von Neumann.

[4]Dice can be considered multi-faced coins.

[5]Dice have existed for thousands of years and are commonly used in games, including high-stakes games.

[6]Marin County, CA, used this method quite successfully after the June 6th, 2006 Primary and the 2006 November 7th, 2006 General Elections.

Figure 2.3. A ten-sided die, tumbler and tray.

Number	*Precinct ID*
0	P_1
1	P_2
$\vdots$	$\vdots$
$N-1$	P_N

Figure 2.4. A numbered list of precincts.

initial proposals involved the use of math (not necessarily complex math). However, the feedback we got indicated the less math, the better. We then considered using computers to perform the math during the selection, but our feedback indicated that no reliance on computers was preferable. We also considered creating worksheets, such as tax worksheets, to guide users through the procedure. However, people generally do not enjoy working on taxes, and if people were to find the worksheets inefficient or frustrating, they might elect to fudge or abandon the procedure.

Keeping that in mind, we tried to separate the "work" from the "sheet" by creating pre-computed lookup tables. The procedure we arrived at requires virtually no math to use.

The basic idea. We begin by preparing a list (numbered $0, 1, 2, \ldots, N-1$) of the population we are sampling from. In California, this would involve preparing a list of the names (or IDs) of every precinct to be included in the audit. Numbering each entry sequentially from 0 to $N-1$ establishes a one-to-one mapping of the integers $[0, N-1]$ to the list of precincts (Figure 2.4). Election officials commit to this ordering before the audit by publishing the list and providing a copy to each political party and each observer. We assume that the audit is not performed until a final electronic tally is available; the goal is to verify whether these alleged election results match the paper records. Election officials commit to the electronic results before the audit by printing the electronic vote totals broken down by precinct, or by writing them onto write-once media (such as CD-ROM), and providing a copy to every interested observer.

Once these preparation steps have been completed, the random selection process begins. In our scheme, an election official rolls an appropriate number of dice to get a random number between 0 and $N - 1$, and then uses the one-to-one mapping established earlier to interpret this number as identifying a single precinct. This precinct is added to the random sample, and the process is repeated until the random sample is of the desired size.

If we used standard six-sided dice, rolling k dice would allow us to randomly choose a number between 0 and $6^k - 1$ by reading off the k outcomes and treating them as a number in base-6. However, requiring election officials and observers to perform base-6 arithmetic may be unreasonable. Therefore, instead of using standard dice, we propose using commonly available ten-sided dice[7], letting each die signify a decimal digit. For example, if we want to select one precinct from a list of 1000, we could use three dice to get a number between 0 and 999.

The above procedure handles the case where N is a power of ten. If N is not a power of ten, one simple method is to roll $\lceil \log_{10} N \rceil$ ten-sided dice[8], and then re-roll if the resulting number is too large. For instance, suppose we want to choose at random from a list of $N = 750$ precincts. Throwing three dice gives us a random number from 0 to 999. Because only the numbers 0 through 749 correspond to precincts on our list, we ignore and re-roll any time we get a number greater than 749. Because every three-digit number has an equal probability of appearing[9], this method produces a uniform distribution over the 750 precincts.

As we have presented it so far, this scheme works reasonably well for a small number of selections (say $N \leq 1000$). However, re-rolling can become a problem. There is a significant difference between using three dice to select 1% of 1,000 precincts, and using four dice to select 1% of 1,001. Below, we address this issue by using more general range-lookup tables.

2.5.1 General dice scheme

As discussed above, re-rolling can become inefficient quite quickly. To address this, a natural optimization is to divide the range of the dice into N equal intervals, letting each interval correspond to a precinct on the list[10].

For example, suppose we want to use four ten-sided dice to select a precinct from a list of $N = 1001$ precincts. First, we divide the range $[0, 9999]$ into 1,001 equal sized intervals $[0, 8]$, $[9, 17]$, ..., $[8991, 8999]$, $[9000, 9008]$. We then let each *interval* correspond to a precinct on our list, allowing the remainder, $[9009, 9999]$, to correspond to a re-roll. As a result, we throw out roughly 1 out of 10 rolls instead of 9 out of 10, improving efficiency greatly. See Algorithm 2.5 for a specification of this scheme.

[7] 10-sided dice can be found at game stores, are often used in board and role-playing games, and typically are numbered 0 through 9.

[8] Here, $\lceil \cdot \rceil$ represents the *ceiling* function (in other words, we round up). $\lceil \log_{10} N \rceil$ just mathematically expresses commonsense: use as few dice as necessary.

[9] Subject to any bias present in the random source. If a *verifiable* and *robust* source is used, according to our requirements in Section 2.3.1, this bias is negligible.

[10] We use division, rather than modulo, because division is a more familiar concept to a general audience.

```
SELECT-PRECINCTS($num\_precincts, sample\_size$)
1  $sample \leftarrow \emptyset$
2  $k \leftarrow \lceil \log_{10} num\_precincts \rceil$
3  $d \leftarrow \lfloor 10^k / num\_precincts \rfloor$
4  while $|S| < sample\_size$
5  do Roll $k$ ten-sided dice to obtain a random $k$-digit number $x$:
6     $x \leftarrow \{0, 1, ..., 9\}^k$
7     $selected\_precinct \leftarrow \lfloor x/d \rfloor$
8     if $selected\_precinct < num\_precincts$ and $selected\_precinct \notin S$
9       then $S \leftarrow S \cup \{selected\_precinct\}$
```

Figure 2.5. General dice scheme.

Numbers	*Precinct ID*
$0 \ldots 8$	P_1
$9 \ldots 17$	P_2
$\vdots$	$\vdots$
$9000 \ldots 9008$	P_{1001}
$9009 \ldots 9999$	RE-ROLL

Figure 2.6. A list of 1001 precincts labeled with equal-sized ranges.

We can simplify the presentation of this scheme by changing how we prepare the numbered list of precincts. Instead of labeling each precinct with an integer, we could label each with its pre-computed range (Figure 2.6). Now, no arithmetic is necessary to do the selection: one can simply roll k dice and then look up the outcome on the list to obtain a precinct.

2.5.2 Analysis

It should first be noted that this simple idea of using range-lookup tables applies to other random sources, and can enable other nice things like allowing for a controlled bias (say to account for precinct sizes).

Advantages. The foremost advantage of this scheme over computer-based PRNGs is the use of an observable random source. Dice are simple, robust and familiar enough to be widely accepted[11]. Dividing up the range of the dice improves efficiency while maintaining simplicity. The efficiency and usability of this scheme is excellent. As shown in Figure 2.7, election officials can easily select a random sample of 1% of the precincts, and even very large counties will not need too many tosses of the dice. For instance, even with $N = 2000$,

[11]The proposal to use dice was met with enthusiasm from election officials in Alameda County.

only about 25 tosses of the dice (on average) are needed to select a random 1% sample. The fairness of the scheme is also verifiable (to the extent that dice are) because the lookup table is published; it can be inspected after the fact to verify that, for example, every precinct was included and had an equal chance of being selected.

Disadvantages. When the size of the sample is a large ratio of the population, many more re-rolls would be expected, as "collisions" occur in the sample selection. For small ratios this loss is negligible.

Unfortunately, the cost of this scheme scales linearly (keeping the sample to population ratio constant) with the size of the sample. While this might be suitable for selecting 1% of 500 or 1,000 precincts, it might be too time consuming to select 1% of 1,000,000.

A second concern is the threat of biased dice. If a malicious party were able to affect the distributions of the dice (say by weighting a die or by swapping in a die with duplicate faces), that party could affect which precincts get selected, or worse, which precincts *do not* get selected.

Efficiency. In an election setting, dice not only have to be rolled, but inspected by observers. This incurs a cost per roll, so we measure the efficiency of our methods by the number of rolls they require.

We can partially reduce this cost by rolling multiple dice at once. If we do this, we must ensure the order of dice is well-specified in advance. One idea is to use dice with different magnitudes on their faces, such as 1's, 10's, and 100's. Another possibility is to use dice of different colors and establish a clear ordering of the colors before rolling the dice. A good choice in the United States would be red, white and blue for their culturally meaningful order. This would enable us to roll three dice (one red, one white, and one blue) at a time; for instance, if the red one comes up 5, the white one 7, and the blue one 2, that would be interpreted as the three-digit number 572.

2.6 Related Work

The idea of using auditing to gain statistical confidence in an election is not new. Many states, such as California, already require random audits or manual tallies in their election codes.

Prior work has looked at the issue of the statistical validity of random audits and what size sample is necessary to achieve a given degree of confidence (60). In these papers, however, the sample selection process is not considered and is assumed to be perfect. Some authors have examined the possibility of auditing individual ballots, stating that if adopted (in a privacy-preserving way), audits of individual ballots could give much higher degrees of confidence with less overall counting (48; 60).

Other people have looked at verifiable methods of sample selection. The IETF uses

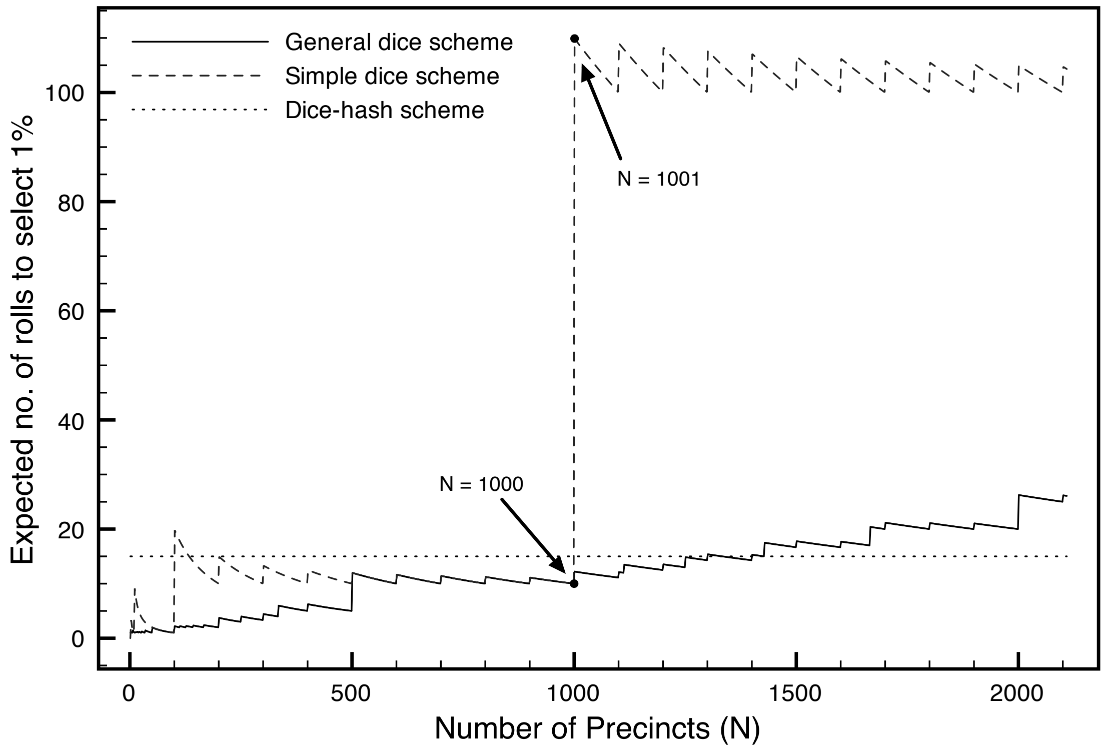


Figure 2.7. The expected number of tosses of 10-sided dice to select a 1% sample using range-lookup tables (General dice scheme) versus simply re-rolling (Simple dice scheme) as a function of population size (N). The Dice-hash scheme is a deterministic function, similar to (26), to calculate a sample selection using a constant number of dice rolls as input.

an algorithm based on a cryptographically strong hash-function, to expand input from a random source into a sample (26). However, the reactions of those in the election community we talked to indicated a resistance to the use of cryptography, particularly from those unfamiliar with hash functions and cryptography, leading us to prefer a dice-only approach.

2.7 Contributions

Sample selection is a critical part of an audit, but has been overlooked in the election setting, as evidenced by the wordings of election code (74), and existing current practices such as the use of software-based PRNGs. We identify non-transparency as a vulnerability for every part of an audit, particularly sample selection, and lay out requirements that a transparent sample selection procedure must meet.

In addition, we present an algorithm and implementation for transparent sample selection, using dice and a printed lookup table. This scheme is simple and relatively efficient, no calculations are needed to perform the selection, and, if adopted, would make elections more trustworthy.

2.8 Conclusion

Transparency plays an important role in the security of election audits. Because auditing is a public process capable of establishing confidence in an election's outcome, it is critical that every part of the process, particularly the random sample selection, be transparent, observable, and understandable to all interested parties. While the current use of computers to generate samples is neither transparent nor observable, we have presented a possible remedy that is simple, verifiable, low-cost, and robust.

Ultimately, it is essential that we choose random audit procedures that are capable of convincing the whole population of the genuineness of the audit, and, in turn, of the election.

Chapter 3

Replayable Voting Machine Audit Logs

Audit logs are an important tool for post-election investigations, in the event of an election dispute or problem. This chapter describes a new approach to logging that we proposed to provide a record of all interactions between each voter and the voting machine. Our audit logs provide a comprehensive, trustworthy, replayable record of essentially everything the voter saw and did in the voting booth, providing investigators a tool for reconstructing voter intent and diagnosing election problems. We show how our design preserves voter anonymity and protects against vote-buying and coercion. We implement a prototype logging subsystem, built on the Pvote voting platform, and demonstrate that the approach is feasible.

3.1 Introduction

Elections do not always go smoothly. When problems occur, or if an election is disputed, it is important that the voting system preserve forensic evidence that can be used to investigate reported problems. Many commercial e-voting systems generate and maintain audit logs—records that describe the operation of a voting machine in an election—for this purpose. Unfortunately, the audit logs in currently deployed voting systems fall short in several respects: they often record only minimal information, omitting vital information that might be needed in a post-election investigation (85); there is not always a clear pattern to what is logged (52); there are few protections to ensure the integrity of the logs, so any failure of the voting software can also corrupt the audit logs (13; 46; 80); and there is no guarantee that the logs contain an accurate or complete record of what happened on election day.

We study how to improve this situation. We show that it is possible to design audit log mechanisms that remedy these shortcomings. Our audit logs are more complete: we expect that in many cases they may help investigators diagnose reported problems, test hypotheses about the cause of those problems, check that votes were recorded and interpreted correctly, reconstruct voter intent (to a certain extent), and possibly even correct problems. For instance, our audit logs provide useful evidence of a voter's interaction with the machine, which may be helpful in reconstructing her intent. Because audit logs can potentially weaken the degree of voter anonymity a voting machine provides, they must be carefully designed to ensure they protect ballot secrecy. We show how careful design of the logging data structures can protect the secrecy of the voter's ballot and protect against vote-buying and coercion.

This work is partially inspired by the second author's involvement in a post-election investigation of the Sarasota Congressional District 13 election (85), where over 13% of DRE ballots recorded no vote in the CD13 contest. In that election, a number of voters alleged that the DRE never gave them a chance to vote in the CD13 contest or that their initial selection in the CD13 contest was not displayed on the DRE summary screen. One of the challenges in that investigation was that, to protect voter privacy, the DREs used in that election did not retain any information that would enable investigators to recreate exactly what voters did or did not see on the DRE screen. As a result, those allegations could only be checked through indirect means. If there had been some way to replay the sequence of screen images that each voter saw and the user interface actions the voter took in response to each screen, this would have enhanced our ability to investigate the cause of the undervote and to ascertain voter intent: for instance, we could have checked whether the CD13 contest was always displayed to every voter, and we could have checked whether there was any evidence that some voters' selections in the CD13 contest were not reflected accurately on the final summary screen. In this chapter, we design a way to retain this kind of information, without violating the secrecy of the ballot.

3.1.1 Problem Statement

We study how the audit logging mechanisms in electronic voting machines should be designed. We want to generate and preserve a comprehensive, trustworthy set of electronic records that can be used to detect and diagnose many kinds of equipment problems and failures. These logs should record useful evidence of voter intent, preserve voter anonymity, and avoid interfering with any other requirements of the voting machine.

We want election investigators to be able to use these audit logs after the election to reconstruct the interaction that each voter had with the voting machine. We focus on designing audit logs that enable investigators to reconstruct everything the voter saw on the voting machine's screen, everything that the voter did (e.g., every location on the screen that the voter touched, every button that the voter pressed), and every ballot that was cast as a result of these actions. Audit logs would not normally be used to count the votes. Instead, the idea is that, in case of problems with the election or election disputes, it should be possible for election investigators to use the logs to reconstruct and infer, as best as possible, the voter's intent. Achieving this goal requires recording far more than today's voting systems.

Ideally, these audit logs would provide an *independent* way for investigators to verify that the voter's intent was recorded accurately, to correct any errors the machine may have made, and to gather evidence and test hypotheses about the possible causes of these errors. In practice, we cannot fully achieve the ideal of full independence: we do not know how to ensure that the audit log mechanism will be truly independent of the voting machine software. However, we seek to minimize the likelihood of failures that simultaneously affect both the records of cast ballots and the audit logs. In particular, we would like to be able to correct or detect many common kinds of failures, such as errors in recording the voter's selections, user interface flaws, ballot design problems, configuration errors, and touchscreen miscalibration.

However, some kinds of failures are out of scope for this research. We assume the software that executes on election day matches the certified version, and that this software is free of malicious logic and backdoors; and we assume that the hardware is trustworthy, correct, and free of tampering. We make no attempt to detect violations of these assumptions, so our audit logs can not solve all problems (especially security issues that involve malicious insiders)—but we hope they will be useful in practice nonetheless.

Audit logs must not compromise ballot secrecy. This poses a significant technical challenge: the more information we record, the greater the risk that this might provide a way to link voters to how they voted, or that it might provide a way for voters to prove how they voted, sell their votes, or be coerced. Many obvious schemes have serious ballot secrecy problems.

We focus on building audit logs for machines that interact directly with a voter, namely, for DREs and electronic ballot markers (EBMs).

It would be useful if these audit log mechanisms could be deployed on existing voting systems, without hardware changes. Because many jurisdictions have recently deployed new e-voting systems, they may be reluctant or unable to replace their equipment any time soon. Therefore, we'd prefer a way to retrofit existing voting systems with better audit log mechanisms simply by upgrading their software. In practice, this restriction places limits on what we can accomplish, so we examine what is the best that can be done simply through software upgrades. We also investigate how future voting system platforms could better support trustworthy audit log mechanisms.

3.1.2 Our Approach

How do we improve the trustworthiness of the logging system? By isolating the logging subsystem from the rest of the voting machine code, we minimize the possible interactions between the logging code and the rest of the voting software. Also, we design the logging code to be as simple and robust as possible, in hopes that this will make it easier to get right, easier to verify, and easier to trust. As we shall see later, the logging subsystem can be dramatically simpler than the rest of the voting software.

How do we make audit logs more useful? Audit logs exist to record evidence of voting machines' operations. We propose that these logs should record the sequence of all I/O from the voting machine *as experienced by the voter*, or as close to that as possible. We record all outputs: if the voting machine has a LCD screen, we will record every image displayed on the screen; if it has a printer, we record all text that is printed. Similarly, we

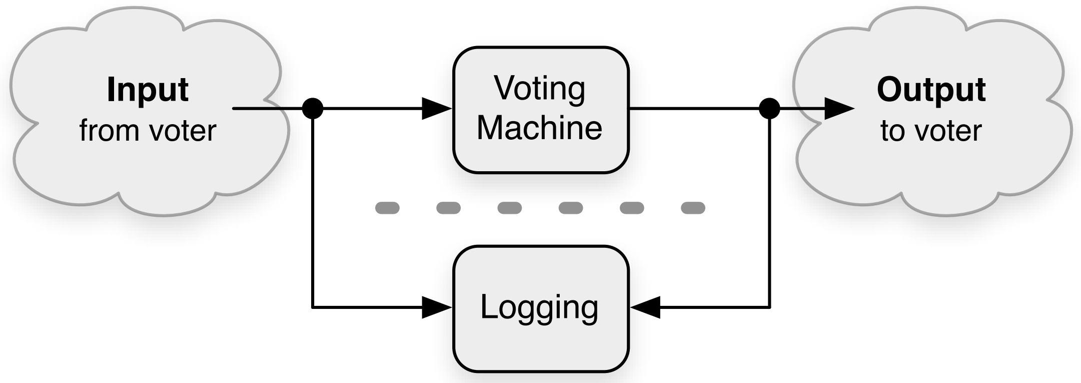


Figure 3.1. Conceptual problem statement. An audit log should record the I/O of a voting system as closely as possible to what the voter experiences.

record all inputs: if the voting machine has a touchscreen input, we record the position of every touch the voter makes; if it has physical buttons, we record every press of these buttons; if it supports other kinds of input (e.g., for accessibility), we record those input events as well. Because this essentially captures all the ways the voting machine can communicate with the voter, and vice versa, and because the results of the voting machine (i.e., the final vote tallies) should be determined by the voters' interaction with the machine through this interface, I/O-based audit logs can capture useful evidence of voter intent.

We store these events in the order they occur, so that it is possible to replay the sequence of all interactions between the voter and the voting machine. We call these types of audit logs *replayable*, because the I/O can be replayed on an independent system without re-implementing any of the logic of the voting machine. As a result, election investigators can use this data to replay anonymized voting sessions and test hypotheses about the election results.

How do we address ballot secrecy? Any system that records votes in the order they are recorded endangers voter anonymity, because it allows an observer who notices the order in which voters cast their ballots and who has access to this electronic record to learn how each voter voted. We protect ballot secrecy by ensuring that the order in which voters vote is independent of the data associated with them. For each voter, we bundle up all of the audit log entries associated with that voter into a single record, so that there is one record per voter—and then we store these records in a random order. Moreover, we are careful to avoid recording the absolute time at which any voter interacts with the voting system or the length of time that it takes for a voter to vote, since these can also breach ballot secrecy.

How do we avoid interfering with the rest of the voting software? We must be sure that adding our logging subsystem to an existing voting machine will not disrupt or interfere with its operation. If the legacy voting software works correctly on its own, then adding our logging subsystem must not cause it to crash, deadlock, misbehave, or otherwise endanger the security, usability, reliability, or privacy of the voting machine. To achieve this goal, we rely upon hardware or software isolation mechanisms (e.g., memory protection or type-safe languages) to prevent the logging code from interfering with the code or state of the rest of the voting software; we expose only a narrow interface between these two software components; we strive to minimize the complexity of the logging subsystem; and we structure our logging code so we can demonstrate that its operations terminate successfully in bounded time.

3.2 Design

3.2.1 Simplicity

In our architecture, the voting machine contains two components: the *voting subsystem* (this includes the legacy voting software; it implements the user interface, voting logic, and vote-recording functionality, among other things) and the *logging subsystem* (which is responsible for storing log records in a private manner). Our logging subsystem presents a minimal interface to the rest of the voting machine. Essentially, we export only an *append record* function[1]. Compared to the complex requirements of a voting machine, this may represent a significant simplification.

3.2.2 Isolation

We assume that we have a trusted platform that provides some mechanism for isolation, and we use it to enforce the separation between the logging system and the voting machine. We have two requirements. First, to ensure that all logged events accurately represent what the voter experienced, the isolation mechanism must protect the logging subsystem in case the voting subsystem misbehaves or runs amok. Second, it must protect the voting subsystem from any negative effects of the logging subsystem. We do not want the introduction of our logging subsystem to reduce the overall security, privacy, or reliability of the voting machine.

Protecting the logging system. The isolation mechanism should prevent the voting subsystem from bypassing or fooling the logging subsystem. In particular, since we are interested in recording the I/O of the system, the logging system should accurately capture all voter-visible I/O, despite any efforts otherwise by the voting machine. This means that we must mediate the voting subsystem's access to I/O devices and ensure that a copy of all inputs and outputs visible to the voter are also sent to the logging subsystem. Also, the logging subsystem needs its own storage medium where the logs can be recorded, and the

[1]Features for reading or deleting log entries should not be available to the voting machine's trusted election-day logic, and hence are omitted from its code base.

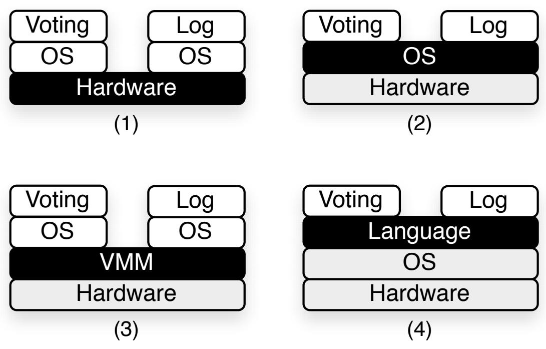


Figure 3.2. Four architectural choices for isolating the voting and logging subsystems from each other.

isolation mechanism must prevent the voting subsystem from reading or writing these logs. Finally, the voting subsystem must not be able to modify the private state or code of the logging subsystem.

Protecting the voting system. The isolation mechanism must prevent the logging subsystem from modifying the private state or the code of the voting subsystem. The voting subsystem needs its own storage medium where electronic cast vote records and other data can be recorded, and we must prevent the logging subsystem from accessing that data.

Ways to implement isolation. We identify four different ways to meet these requirements (see Figure 3.2):

1. We could use hardware isolation, e.g., by running the voting subsystem on one microprocessor and the logging subsystem on another microprocessor and restricting their connectivity to each other and to I/O devices (70).

   The Prime III voting system works similarly. It physically routes the video and audio output of the voting machine through a VHS recorder (82; 21), and the video tape serves as a record or log of the voting session. Although now the physical security of the recorder and tapes must be trusted, this provides strong isolation. A misbehaving—even maliciously programmed—voting machine will have to struggle to avert the logging mechanism. Voter anonymity, however, remains an issue in Prime III, as we discuss below and in Section 3.5.

2. We could use OS isolation mechanisms. For instance, we could run the voting subsystem and the logging subsystem in two separate processes and rely upon OS memory protection to keep them separate.

3. We could use a virtual machine monitor, running the voting subsystem and logging subsystem in two different virtual machines (25). This allows us to mediate all of

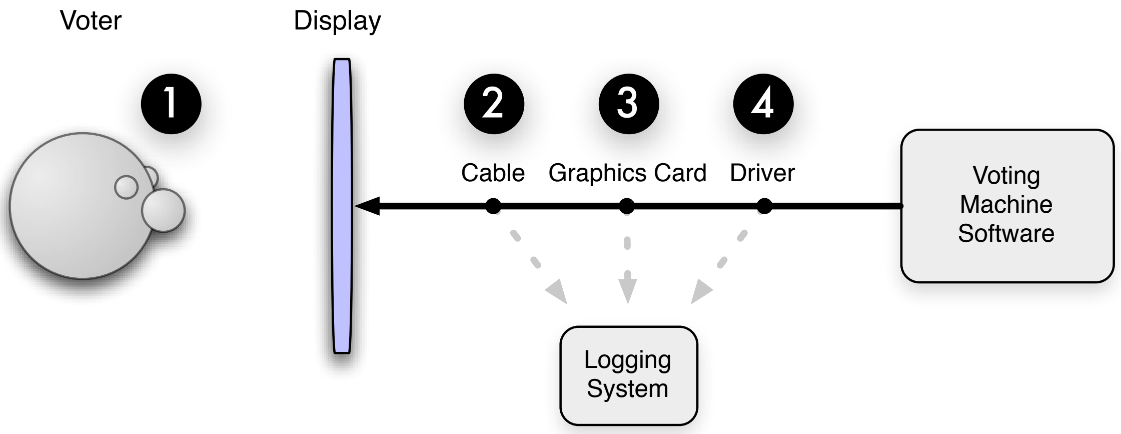


Figure 3.3. Four possible places where the video output signal could be recorded. All transformations to the video signal that occur after it is recorded and before it is seen by the voter can cause undetected inaccuracies in the audit log and hence must be trusted.

their interactions with each other and with I/O devices using a small module integrated into the VMM (34).

4. We could use a type- and memory-safe programming language for isolation, relying upon these language features to ensure that one subsystem cannot tamper with the private state or code of the other subsystem.

These approaches have different tradeoffs. For instance, hardware isolation may provide the greatest level of assurance, but it would require hardware changes to existing voting machines. Software isolation may be easier to retrofit, but it requires trust in the OS kernel, VMM, or language platform.

We note that the isolation mechanism does not need to be perfect to be useful. Even a suboptimal isolation mechanism can still be useful for detecting prevalent classes of important failures, like configuration or operator errors. Experience suggests that, for many election administrators, these problems are more palpable and urgent threats than the adversarial threats computer security experts often take for granted.

3.2.3 Crafting a replayable audit log

Capturing voter intent. As explained in Section 3.1.2, we are primarily interested in recording the I/O of the voting machine, *as experienced by the voter*. However, the extent to which we can do this is limited by where in the data path the I/O is recorded. Consider the example of video output. Figure 3.3 shows four places where the video signal could be recorded. With the exception of perhaps an external camera mounted near the eye-level of the voter (1), the video that is recorded will be transformed before the voter sees it. For instance, recording the analog video output from the video card (2), the digital bitmap rendered by the graphics card (3), or the screen images produced by the graphics device driver (4) would not differentiate between an image displayed on a clean screen, or one displayed on an obscured or damaged screen. Similarly, a signal recorded from the graphics card frame buffer (3) may not detect a disconnected cable or a fault in the hardware D/A converter, though it *could* detect bugs in the device driver or voting software that cause the wrong image to be displayed.

Voter anonymity. As one might imagine, what gets logged strongly affects the degree of voter anonymity of the system. Even without recording the voter's name, face, voice or any other identifying information, data from the voter's *interaction with the voting machine* may still be enough to identify her. For instance, if we store a timestamp with each log entry, and if the log entry can be linked to a record of the voter's votes, then this may be enough to link a voter with how they voted. This is especially unfortunate since common practices—and even the Voluntary Voting System Guidelines (6)—suggest timestamping all log entries. We protect voter anonymity by never associating timestamps with our log entries. A voting machine is always free to keep a separate supplemental log where all entries are timestamped as long as those entries reveal nothing about how the voter voted; we consider that out of scope for this research.

More subtly, duration information can compromise voter anonymity. For instance, if each log entry records the amount of time it took for the voter to vote, and if each log entry can be linked to a record of that voter's votes, then an observer who notes how long it took for Alice to vote may be able to infer how Alice voted. Even partial information about the time the voter spends on a particular screen could potentially violate voter anonymity. We counter this threat by never storing duration information: the log record associated with a voter contains the sequence of screen images and input events but no information about the duration between these events. This does leave one residual risk: if the number of events is correlated to the time it takes for a voter to vote, this may endanger voter anonymity. We leave it to open work to evaluate this issue more thoroughly.

It is impossible to protect voter anonymity if the voting subsystem is malicious or buggy. Therefore, our anonymity goals are conditioned on the assumption that the voting subsystem is correct, non-malicious, and does not itself violate voter anonymity.

We address vote-buying and coercion in Section 3.2.5.

Video. Video—the sequence of screen images shown on the voting machine's display—is arguably the machine's primary channel of communicating with the voter. Ideally we would like to record this video in real time, so that election investigators can replay a "movie" of the video images that the voter saw. However, real-time video reveals vote duration, which is a problem for voter anonymity. Instead of recording real-time video, we record a frame of video only when the image on the screen changes. Because voting machines typically show static content that changes only in response to user input events—for instance, they normally do not display animations—this approach redacts timing information by concealing how long the voter spends on each screen.

This approach essentially consists of run-length encoding the video, and then omitting the run lengths. Unfortunately, this approach does remove some information that may be relevant to election investigators: for instance, if investigators are concerned that the voting machine took too long to respond to user inputs, our audit logs will not contain enough information for them to investigate this hypothesis.

We assume that the voting subsystem does not display the current time to the voter on the screen. Fortunately, it should be straightforward to modify legacy voting software to ensure that this is the case.

We do not record audio. Real-time audio recording has the same problem with duration as video. However, unlike a sequence of static screen images, audio output is inherently temporal. We have not found any clean solution to this problem. Consequently, in our design, we do not record audio outputs from the voting machine. We recognize that this may omit important information that would be useful in an election investigation, and we consider it an interesting open problem to eliminate this limitation.

Other issues. Other subtle issues can threaten voter anonymity. First, *input device preferences* may be correlated with voter identity, especially for less common accessibility devices. For instance, if an observer notices that Alice is the only voter to vote using a sip-and-puff input device, and if the input modality is recorded in the audit log, then it will be possible to identify the log entry that corresponds to Alice and hence learn how she voted. We currently do not have a good solution to this problem, so our approach currently fails to support accessibility.

Second, *externally controlled signals* might mark or identify audit logs. For instance, if the voting machine accepts speech input (presumably through voice recognition), external sounds or noise may be reflected in the audit log entries and thus may endanger voter privacy. We do not support this kind of voting interface; we assume that the user interface is such that all input events come from the voter, not from the environment.

3.2.4 Storing the logs anonymously

The way we store the logs—the data structure we use and the physical medium on which it is stored—also affects voter anonymity. To avoid revealing information about the order in which log entries were inserted into the log, we require the log storage unit to be *history independent* (59; 40; 58). See Section 3.3.4 for the data structure and algorithms we use.

3.2.5 Vote-buying and coercion

It is important that we *prevent* voters from proving how they voted, even in the case where voters are colluding with a vote-buyer or coercer. While we cannot completely eliminate these risks, we can provide reasonable protection. In particular, we require that the logging subsystem avoid making the problem any worse than it already is with existing voting systems.

We only attempt to prevent vote-buying and coercion under the following threat model. We assume that the voting machine and all voting software is non-malicious (for a malicious voting machine can easily enable vote-buying). We are not concerned about attacks where the vote-buyer or coercer are in collusion with an election official or other insider. There are already many ways that a voter can prove how she voted to her election official: for instance, she can enter a pre-arranged unique string as a write-in in some contest, or she can enter an unusual pattern of votes in down-ballot races. Our audit logs do provide additional ways to do so, for instance by entering a special pre-arranged sequence of "next screen"/"previous screen" inputs, but this does not provide the voter with any extra power she does not already have. For these reasons, we assume that all election officials and other insiders are trusted not to cooperate in vote-buying or coercion schemes. Instead, we focus only on preventing vote-buying and coercion by outsiders, and we attempt to ensure that voters cannot prove how they voted to any outsider.

Despite the possibility of voter collusion, an audit logging system still must provide anonymity to the remaining voters. In other words, if a voter is not intentionally colluding *with an insider* to prove how they voted, we do not want their ballot or audit log to be linkable to their identity. We acknowledge that the increased information we record about a voting session reveals strictly more information that could be used to link it to a voter. For instance, an elderly voter with known poor motor reflexes may have a distinct pattern of voting that may indirectly appear in the audit logs. The extent to which this is a problem is a subject for future work, but we point out that this kind of information already exists in other forms of voting, such as the pressure and method of filling in bubbles, or the handwriting for a write-in candidate.

Ideally, we would prefer if all audit logs could be routinely released to the public, to enable any interested party to perform their own analysis on these logs. Unfortunately, a policy of releasing our audit logs to the public after every election introduces vote-buying and coercion risks.

One way to prevent vote-buying and coercion would be to treat our audit logs as privileged (not public) information. Under this model, election officials would protect the confidentiality of these logs and avoid disclosing them to untrusted individuals. In particular, audit logs would not be released to the public. This would suffice to ensure that voters cannot use the audit logs to mark their ballots and prove to an outsider how they voted, eliminating the incentive for vote-buyers or coercers to try to pressure voters. This strategy mirrors the requirement that electronic cast vote records must not be released in raw form to the public (lest they enable vote-buying or coercion via pattern voting or special write-ins); in other words, in this model, audit logs would be subject to the same confidentiality requirements as existing cast vote records. Somewhat surprisingly, this is sufficient to ensure that our audit logs do not make vote-buying and coercion any easier. In this model, our scheme would not harm the transparency of the voting system, but neither would it improve

```
INIT()
 1  $session\_id \leftarrow \{0,1\}^{128}$ uniformly at random
 2  $sequence\_id \leftarrow 0$

LOG($s_1, s_2, ..., s_k$)
 1  $s \leftarrow$ SERIALIZE($session\_id, sequence\_id, s_1, s_2, ..., s_k$)
 2  $sequence\_id \leftarrow sequence\_id + 1$
 3  ADD-RECORD($s$)
```

Figure 3.4. Algorithms to initialize a log record, and append entries to it, utilizing the ADD-RECORD($\cdot$) method of the HIDS (see Figure 3.9). SERIALIZE may use any method for serializing its inputs to a uniquely decodable string, e.g., using length-prepending.

transparency: the logs would allow election officials to investigate election results, but not allow members of the public to perform their own investigation.

Another possibility would be to accept some risk of vote-buying and coercion in exchange for better transparency. For instance, officials might ordinarily treat audit logs as privileged, except that in the special case of a contested election, the logs might be released to the independent auditors, candidates, and their representatives. Ultimatey, the extent to which audit logs should be kept private depends on many factors, including the risk of vote buying and coercion in any particular region, so the handling of our audit logs may be a policy matter that is best left to local election officials who are familiar with local conditions.

3.3 Implementation

We prototyped our design by extending Pvote (86), a voting system written in Python. Pvote is already remarkably compact because of its use of a *pre-rendered* user interface. Unfortunately, a security review by five computer security experts suggested that, even for a system as small as Pvote, it is still difficult to be confident that voting software is free of bugs (87).

3.3.1 Isolation

Pvote is written in Pthin, a subset of Python that includes only a small number of language primitives. Our prototype relies on the memory- and type-safety of Pthin for isolation. In particular, this makes it easy to verify that no function in Pvote can access or modify any part of the logging component's memory or instructions except through the logger's public interface, and vice versa. This interface supports only three publicly

	Election-day mission-critical functions	Supporting functions
`Log`	*Init. & logging:* 23 lines	*Reconstructing records:* 41 lines
`HIDS`	*Adding records:* 87 lines	*Listing records:* 39 + *Resetting:* 20 lines
Total	**110 lines**	**100 lines**

Figure 3.5. Lines of Python code, as counted by `sloccount`(81).

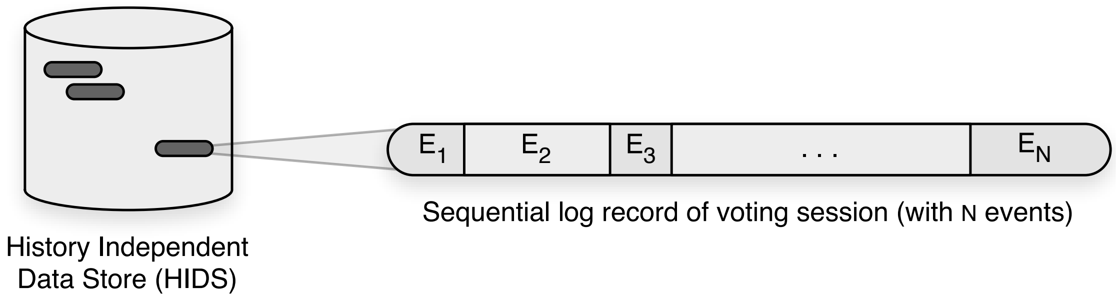


Figure 3.6. A conceptual view of logging. A log record consists of an ordered sequence of log events. The log records are stored in the HIDS in a random order.

accessible operations[2]: INIT(), which begins a new log record for a voting session; LOG(·), which appends an event to the record; and COMPRESS(·), an optional helper function we discuss in Section 3.3.3. Algorithms for the first two operations are described in Figure 3.4.

3.3.2 The logging subsystem

We implemented the logging subsystem in 110 lines of Pthin code. An additional 100 lines of code support reading and reconstructing the logs so they can be replayed after the election (see Figure 3.5).

In our system, a log is a set of log records, each corresponding to a single voter. A log record is an ordered sequence of events. Each event is an ordered sequence of arbitrary-length byte strings. (See Figure 3.6.) Thus, we execute INIT() once for each voter at the beginning of their voting session to create a new log record, and we invoke LOG(·) at every voter-visible I/O event to append the event to that log record. It is up to the programmer to log events consistently. Fortunately, we have found that it is easy to manually inspect the code to verify that the logging functions are called at every relevant place in the Pvote code.

In our implementation, the first string of an event represents the type of event that occurred. The remaining parameters include data for that event. The events we log in our

[2]Python does not itself distinguish between public and private or protected methods, but we found it easy to verify by inspection of the code that only the "public" methods are ever called within Pvote.

Event type	Parameters	Explanation
`display`	*bitmap, width, height*	Records the raw RGB image whenever display changes
`touch`	*x,y*	Logs coordinates whenever the voter presses the screen
`key-ress`	*key*	Logs any keypad button that was pressed
`print`	*text*	Records data sent to ballot printer
`ballot-definition-read`	*hash*	When ballot-definition is read, records its SHA-1 hash
`reset-button`	-	Indicates a voting session is complete and will reset

Figure 3.7. The six Pvote I/O events our system logs. For each event occurrence, we log the corresponding parameters. From the sequence of these recorded events we are able to replay the most important aspects of a voting session.

prototype are summarized in Figure 3.7. For instance, the place in the code that reads a touch event calls

```
Log.log( "touch", x, y )
```

where `x` and `y` are string representations of the integer coordinates (pixel offsets) where the screen was touched. Likewise, whenever the screen is updated, we record the new image that will be displayed on the screen, in raw bitmap form.

Video. As we discuss in Section 3.2.3, we do not record real-time video. Instead we record a screenshot of the display whenever the display image changes. Our implementation uses the Pygame (63) library to record the bitmap displayed to the voter. In particular, in the one place in the Pvote code where the display update method is called, we log the raw RGB pixel data of the complete displayed image.

3.3.3 Compression

Recording bitmap images can be memory and bandwidth intensive. For better efficiency, we experimented with compressing the bitmap data before logging it. We were interested in capturing this data exactly, so we considered two lossless compression schemes: run-length encoding, an encoding that replaces runs of identical bytes with a single byte followed by the length of the run; and the standard `zlib` encoding. For run-length encoding, we tested implementations in Python and C. For the zlib encoding, we used the standard zlib library of Python (written in C). While we do not rule out other forms of compression, we found run-length encoding particularly appealing for its simplicity, and its suitability for compressing screenshots containing large regions of the same color.

A drawback of compression is the complexity it adds to the logging subsystem. Libraries written in C additionally lack the memory- and type-safety properties of a pure Python implementation. In this case, we must treat compression as a trusted operation, another reason to prefer a simple algorithm like run-length encoding. Despite the draw-

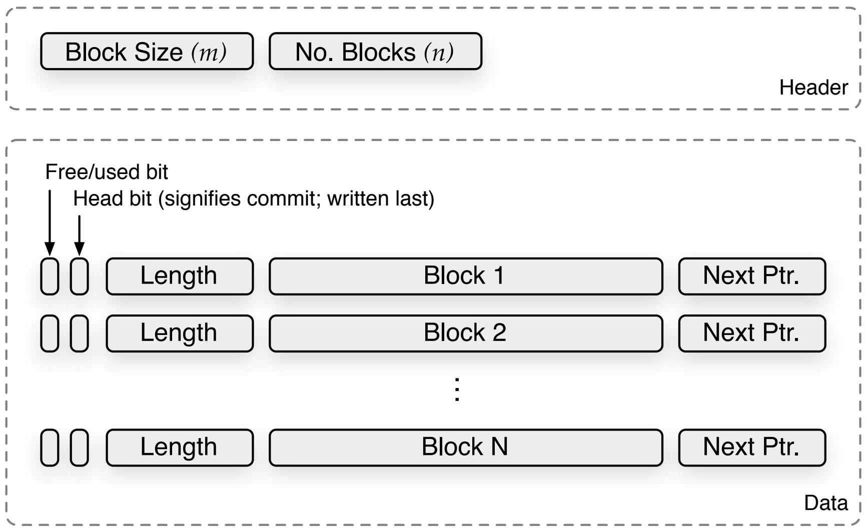


Figure 3.8. The basic data structure of the History Independent Data Structure (HIDS) used by our logging system. This structure is intended to map well to NAND flash memory, where each row of the data section maps to a page in flash.

backs, our evaluation of compression found it very useful for improving the performance of logging video keyframes in our prototype (see Section 3.4).

3.3.4 History independent data structure

As we explain in Section 3.2.4, the order of the voting session log records must be anonymized. We design our log data format to be history independent:

> "A data structure implementation is *history independent* if any two sequences S_1 and S_2 that yield the same content induce the same distribution on the memory representation." (59)

We need our history independent data structure (HIDS) to provide an efficient insertion operation; support for large, variable-length records; and a simple implementation. However, the last requirement, *simplicity*, is our primary objective because we want to make the code easy to verify and to gain confidence in.

The HIDS we implement is shown in Figure 3.8. Given a block size m and the total number of blocks n, we initialize the data section of our HIDS to the known default value of the underlying memory (e.g., 0). To ensure that insertions will not fail, we require the HIDS to allocate a constant factor c more space than it expects to use. For expository purposes we assume $c = 2$. The data structure is a table of blocks plus metadata, which may be linked to other blocks to form a list. The HIDS is used to store the set of log records, and each record is inserted randomly into this structure as a linked list of blocks. By design, the only

```
ADD-RECORD($record$)
 1  $b \leftarrow \lceil \text{LENGTH}(record)/m \rceil$
 2  $data \leftarrow$ Split $record$ into list of $m$-sized blocks
 3  for $i \leftarrow 0$ to $b - 1$
 4  do $loc[i] \leftarrow$ CHOOSE-FREE-BLOCK()
 5
 6  for $i \leftarrow 0$ to $b - 1$
 7  do WRITE-SPECIFIED-BLOCK($i$)
 8
 9  $j \leftarrow loc[0]$
10  $length_j \leftarrow \text{LENGTH}(record)$
11  $head\_bit_j \leftarrow True$

WRITE-SPECIFIED-BLOCK($i$)
 1  $j \leftarrow loc[i]$
 2  $used\_bit_j \leftarrow True$
 3  $block_j \leftarrow data[i]$
 4  if $i < b - 1$
 5     then $next\_ptr_j \leftarrow loc[i + 1]$

CHOOSE-FREE-BLOCK()
 1  for $i \leftarrow 0$ to $127$
 2  do $r \leftarrow \{0, 1, ..., n - 1\}$ uniformly at random
 3       if not $used\_bit_r$
 4          then return $r$
 5  error "fail gracefully"
```

Figure 3.9. Algorithm for inserting records into our history independent data structure (HIDS). We assume the table is a constant factor larger than the maximum expected size, to keep the data structure sparse and allow for fast, simple insertion.

operations this data structure supports are ADD-RECORD($\cdot$), which adds a record randomly to the HIDS (see Figure 3.9), and LIST-RECORDS(), which reconstructs the contents of all the records in a random order. The latter operation is only done to retrieve the data after an election, so its performance is not as important. Deletion and lookup of records are intentionally not supported.

To insert records, we use a cryptographically-strong pseudo-random number generator (PRNG). Although a hash-based approach is also viable—and there are reasons it may be appealing[3]—we choose to work with a PRNG to minimize the complexity of the code.

Theorem 3.3.1. *Our HIDS is history independent.*

[3]Particularly, a hash-based approach with a unique, deterministic ordering of the records avoids possible subliminal channels (58).

Proof. Our data structure is a table of blocks. Each record insertion writes data to one or more unused blocks. Each block is selected independently and uniformly at random from the remaining unused blocks. Suppose we insert r records (comprising k blocks) into a data structure of n total blocks. Let σ denote the state of the resulting data structure after these insertions. The probability of seeing the memory representation σ after these r insertions is

$$\Pr[\sigma] = \frac{1}{n} \times \frac{1}{n-1} \times \frac{1}{n-2} \times \cdots \times \frac{1}{n-(k-1)},$$

independent of the order in which in these records were inserted. Therefore, the data structure is history independent. $\square$

Theorem 3.3.2. *If we allocate twice as much space as required ($c = 2$),* ADD-RECORD$(\cdot)$ *will fail with probability at most* 2^{-128}.

Proof. Assume that our HIDS contains space for n blocks of data, and that we insert records containing at most $n/2$ blocks in aggregate. At any time ADD-RECORD$(\cdot)$ is called, at least $n/2$ blocks are available, so the probability of selecting a non-empty block in lines 2–4 of CHOOSE-FREE-BLOCK$()$ is $p \leq 0.5$. The probability of failing to find a free block after 128 independent trials is $p^{128} \leq 2^{-128}$. $\square$

We designed this data structure to be resilient in the face of hardware failures. In particular, consider a fail-stop model where at any point the storage device can fail, causing all subsequent reads and writes to fail with an error. We assume that all small writes are atomic and synchronous, with no re-ordering of reads or writes. As a reminder, we assume that the data storage device is originally initialized to a default value (e.g., 0), different from the encoding of $True$. With these conditions, we can prove:

Theorem 3.3.3. *Under this failure model, if* ADD-RECORD$(\cdot)$ *fails while trying to insert a record, the data structure will not be corrupted. Also, any partially inserted record(s) can be distinguished from complete records.*

Proof. The first thing that gets written to a block is its $used\_bit$, which is flipped to $True$ in line 2 of WRITE-SPECIFIED-BLOCK$(\cdot)$. In the failure model outlined above, this happens atomically. Once this bit is flipped, this block (even if not completely written) will not be selected by CHOOSE-FREE-BLOCK$()$ again. Moreover, since we use the underlying storage medium in a write-once fashion, any failure that may occur during writing of this log record will not affect the other log records that have previously been successfully inserted into the data structure.

What we need to show then is that an incomplete block, or a block that is part of an incomplete record (i.e., part of a linked list of blocks that was never committed), will be distinguishable from a block from a complete record. The final line of ADD-RECORD$(\cdot)$ sets $head\_bit_j \leftarrow True$ for the first block of the record. Under the assumptions given earlier, this happens atomically, after the previous operations have completed. A complete record is therefore recognizable because it starts with a "head block", i.e., a block with $head\_bit$ set. Blocks that are part of an incomplete record (including incomplete blocks) will be "head-less" and therefore distinguishable. $\square$

Physical storage. The history independence of the device that stores our HIDS must also be considered. For instance, even though our data structure is write-once, the file system on which it is stored may for its own reasons decide to rearrange the physical memory, while maintaining the appearance of a consistent and stationary logical address space. In fact, file systems optimized for flash, our target storage medium, will do this for the sake of reliability and performance (32). We are not aware of any existing electronic voting system that takes this into account. For example, the Sequoia voting system attempts a history independent data structure for storing vote records. The storage device, however, is a commodity removable Compact Flash card running a standard file system not designed for history independence (13).

Our data structure is designed so it can be stored directly on NAND flash memory, in lieu of a file system, bypassing the flash translation layer (FTL) normally responsible for mapping logical addresses to physical ones to provide wear-leveling and other useful functions. We discuss below how we compensate for this.

Our data structure maps onto NAND flash in the following way. Each row of the data section (containing $used\_bit_j$, $head\_bit_j$, $length_j$, $block_j$, and $next\_ptr_j$) will be stored in its own page in flash (the block size m would be chosen accordingly). To make this work well, we have designed the HIDS to meet the following requirements: the data structure should modify storage in a write-once fashion (since re-writing is slow and creates the possibility that failures could corrupt data structure invariants); the data structure should take into account the reduced reliability of writing data in flash (the flash translation layer normally accounts for this with bad block mapping techniques (32)); and the data structure should account for wear-leveling (again the flash translation layer normally accounts for this).

It is easy to verify that our data structure is write-once in the sense that a bit that has been programmed to 1 or 0 is never modified. However, NAND flash is usually programmed a page at a time. By writing the first block of a record last (along with its $head\_bit$), we get a data structure that is write-once with page granularity.

Writing directly to flash exposes us to common reliability problems normally handled by the flash translation layer. We can make up for this by keeping a list of bad blocks to avoid[4] that is fixed and static for the duration of an election, and by using error correcting codes (ECC). NAND flash usually provides a little extra memory per page specifically for ECC. Instead of maintaining a dynamic bad block map, and moving blocks around when they fail[5], we can use ECC to detect and correct single-bit errors in place. After the election, we recover the original stored values and update the static bad block list.

The final common function of the flash translation layer is to perform wear-leveling, a function that ensures the distribution of writes is spread out over many blocks and not

[4]Defective blocks in NAND flash are typically identified at the time of manufacturing. Blocks may also fail in operation, for which a bad block map is usually maintained.

[5]Dynamic bad block mapping would typically copy bad blocks to new locations when an error is detected and update the bad block map. However, either of these actions could unwittingly reveal information about the history of the data structure.

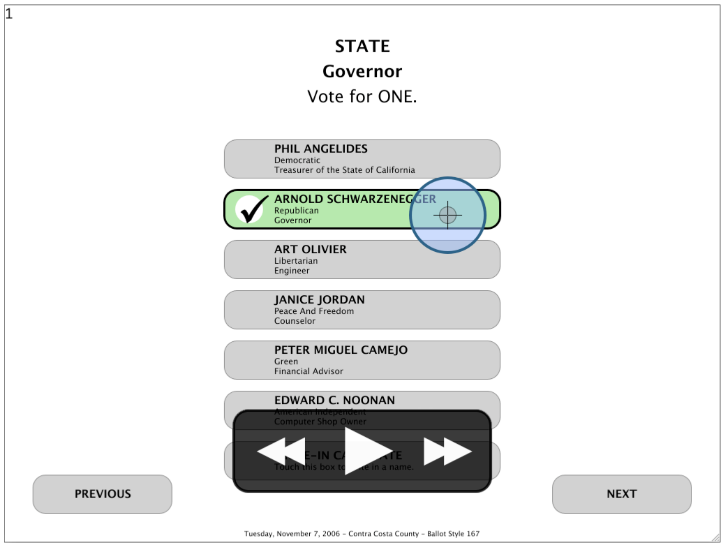


Figure 3.10. A screenshot of our replay mechanism showing a selection screen previously shown to a voter. The cross-hair indicates the location of a touch event in the box for Arnold Schwarzenegger.

concentrated on a few. Because our data structure is write-once, and writes are uniformly distributed over its memory space, our data structure implicitly wear-levels.

3.3.5 Authentication and integrity of the log

There is an orthogonal—but important—issue to our discussion of trustworthy replayable audit logs, and that is ensuring that the logs cannot be tampered with and that they are authentic. We consider this out of the scope of our research, though we point out two possible approaches. First, one could imagine the voting machine digitally signing the entire data structure following the election. This would not protect against mid-day attacks, but it could help reduce chain-of-custody concerns. Alternatively, one could use history-hiding, append only signatures (12) to sign each block of data as it is entered into the HIDS, limiting the extent of a mid-day attack.

3.3.6 Replay Mechanism

The recorded I/O should be descriptive enough that it is possible to replay the logs without any knowledge of the voting machine's design and without re-implementing any of its logic. Our implementation is similar to a video player, in that the user may select any one of the voting sessions (not knowing when the session took place) and step or play through all the events (see Figures 3.10 and 3.11). The replay program then renders images on the screen and displays the location of every touch on the touchscreen.

```
SWITCHING TO RECORD 2 of 3.
Record 2 has 62 log entries.
0  ['hash', '\x9f\x81eJ\x05\xa9\xe8QN\xe7}#...']
1  ['display_changed', 'x\x9c\xec\xddu\xb8\x15...', '1024', '768']
2  ['key', '306']
3  ['key', '306']
4  ['touch', '382', '220']
5  ['touch', '573', '596']
6  ['touch', '805', '692']
7  ['display_changed', 'x\x9c\xec\x9du\x9cU...', '1024', '768']
8  ['touch', '90', '682']
9  ['display_changed', 'x\x9c\xec\xddu\xb8...', '1024', '768']
10 ['touch', '797', '723']
11 ['display_changed', 'x\x9c\xec\x9du\x9c...', '1024', '768']
12 ['key', '54']
13 ['touch', '621', '575']
14 ['touch', '857', '708']
15 ['display_changed', 'x\x9c\xec\xddyx\x14...', '1024', '768']
16 ['touch', '634', '300']
17 ['display_changed', 'x\x9c\xec\xdd\x07X...', '1024', '768']
18 ['touch', '624', '370']
⋮
```

Figure 3.11. An example of recorded events that can later be replayed. Within a `record` (a voting session), the `entries` are sequential. However, the order of records is independent of the actual order of voters. Note that the `display_changed` entry includes as a parameter a serialized bitmap image (truncated in the figure) of the entire screen as shown to the voter.

3.4 Evaluation

3.4.1 Setup

We evaluated our implementation on a ballot definition based on the November 7, 2006 General Election of Contra Costa County, California[6]. This example ballot definition included only five contests from that election. The five contests are presented to the voter on 12 distinct screens, linked by "next" and "previous" buttons. There are also separate screens for entering write-in candidates.

We use a 1GB history independent data structure, allocated as 524,288 blocks, each 2KB long. We log the events shown in Figure 3.7, and we compress the video images using run-length encoding. We used a 1.83 GHz Intel Core Duo processor MacBook Pro running Mac OS 10.5 with 1.5GB 667MHz DDR2 SDRAM and an 80GB Toshiba Serial-ATA hard drive for our tests.

3.4.2 Performance measurements

We tested the performance of our prototype implementation on this ballot definition by casting a number of test votes using our code. We have not conducted a formal user study, but we expect these figures to be useful indicators of the viability of our approach.

[6]This ballot definition is included with version 1.0b of the Pvote source code.

Average time to log one image

	CPU time to compress	Estimated insertion time	Resulting size after compression		Lines of C code to implement	Total estimated time
No compression	0 ms	384 ms	2,304 KB	100.00%	**0 lines**	**384 ms**
Zlib compression	73 ms	8 ms	44 KB	1.92%	**7734 lines**	**81 ms**
Run-length enc.	16 ms	19 ms	116 KB	5.05%	**101 lines**	**36 ms**

Figure 3.12. A comparison of average latencies incurred by compressing and storing one screenshot image in our prototype implementation. Images are 1024x768 pixel RGB images, about 2MB each uncompressed. The total time to log an image is estimated as the sum of the CPU time to compress and the time to insert the compressed image into the HIDS. The estimated insertion time is a function of the number of random-access reads and writes required to insert the image data into the HIDS. For evaluation we assume a cost of 25μs for random-access reads and 300μs for random-access programming to a 2KB block, representative of what might be expected in a NAND flash implementation. The third column shows the average size of a compressed screenshot, and the fourth column shows the compression ratio of the compression scheme. The number of lines of C code required to implement compression is also shown to give the reader a sense of the relative complexity of compression.

For this ballot definition, our system records on the order of 100 separate I/O events per voting session. These events occupy on the order of 1500–2000 blocks in the HIDS (using run-length encoding), which amounts to 3–4MB of flash memory used per voting session. The majority of this data is image data because every change on the screen requires us to record a 2MB screenshot of the display. As a result, although most events use only a small fraction of the 2KB block they are written to, the total space consumption is dominated by the display events.

These measurements can be used to estimate the amount of flash memory that will be needed, as a function of the number of votes cast. We assume that the size of the storage medium will be conservatively provisioned to ensure that the HIDS does not become more than 50% full. With these parameters, a 1GB HIDS can accommodate approximately 150 voters without exceeding the 50% capacity limit, for an election with a similar number of contests as the one we tested. Our experience is that it is rare for a single DRE to process more than 100–150 voters on election day, due to limits on how quickly voters can make it through the voting process. Consequently, we expect that 1GB of flash memory should suffice for an election with a similar, five-contest ballot definition.

We also measured the latency introduced by our logging subsystem, to evaluate whether it would affect the responsiveness of the user interface. Our measurements showed that the total latency is dominated by the time it takes to store screenshots, which even after compression are 50–100 times the size of other logged events. We found that the time to store a screenshot in the audit log can be attributed almost entirely to two factors: the CPU time to compress the raw bitmap image, and the time to read and write blocks on non-volatile storage. We measured the CPU time for compressing screenshots on our implementation; results are shown in the first column of Figure 3.12. Also, we found that inserting a RLE-compressed screenshot to the audit log requires about 65 random-access page reads and writes to flash memory, on average. Based on an expected latency of 25μs per random-access read and 300μs for programming a 2KB page, we estimated the flash-related latency that our implementation would incur. The second column of Figure 3.12 shows these flash latency estimates for each compression method[7], and the last column shows our prediction of the total latency introduced by our logging subsystem per user interface event. For instance, when using RLE compression, we expect that our logging subsystem will introduce less than 40ms of latency per user interface event. We do not expect this to noticeably affect the responsiveness of the voting machine's user interface.

To cross-check these estimates, we also validated these latency estimates by measuring the performance of our implementation on a laptop, using a hard disk instead of a flash device for nonvolatile storage. Of course, seeks are much slower on a hard disk, so one would expect this to be appreciably slower as a result. After correcting for the difference in seek times, the measured performance on our laptop is consistent with our estimates in Figure 3.12.

3.4.3 Practicality and cost

Our experimental setup includes only five contests. The actual election had 115 different contests with on the order of 40 contests shown to any single voter, or about eight times the size of our test ballot. If we extrapolate proportionally from the performance estimates above we would expect to need 8GB of storage for a machine servicing fewer than 150 voters. If we further assume NAND flash costs \$10 per GB, this comes out to a memory cost of \$80 per machine or around 53 cents per voter. Because the memory is reusable, this cost could be amortized over multiple elections. While this is just a rough estimate, it indicates that the cost of this scheme is not outright prohibitive. Given the declining cost of storage, we expect that replayable audit logs will become more affordable in the future.

While thinking of ways to reduce costs, we considered recording the complete voting sessions of only a random sample of voters. While the resulting audit logs could certainly be useful to detect and analyze many threats and failure modes, this comes at the expense of a complete picture of the election day events. There is a qualitative difference between explaining an argument to the public based on complete data, versus an equally strong argument based on statistics.

3.4.4 Discussion of goals

We achieve a usable, replayable logging system in a small number of lines of code. The algorithms chosen—and our design choices in general—prioritize simplicity over ef-

[7]These estimates are measured in the best case, when the HIDS is empty. In worst-case conditions, where the HIDS is up to 50% full, the number of reads will increase by at most a factor of $2\times$, while all other contributions to latency remain unchanged.

ficiency and features, for the purpose of increasing the trustworthiness of the code and design. Our use of language-based isolation, however, is suboptimal even though it may still be useful for a large class of problems, particularly those involving misconfiguration or operator error. Hardware-based isolation, on the other hand, could result in a strong replayable auditing system.

Our system records all the relevant I/O Pvote provides to a voter except audio. We leave audio to future work because of our concern with duration and anonymity. Our approach of recording only screen images results in data that captures useful evidence of voter intent, while removing precise duration information. It is less clear how to do something similar for audio, because audio is inherently temporal.

3.5 Related Work

The two most closely related works we know of are Prime III (21) and the independent audit framework of Garera et al. (34). Prime III is a voting system especially designed to accommodate voters with visual or aural impairments. In Prime III, all video and audio from the voting machine is copied to a VHS or DV recording device as it is displayed to the voter. This ensures that the audit logs are independent of the proper functioning of the software in the voting machine. However, Prime III's electronic records reveal significant information about the time, order and duration of votes cast and thus endanger voter anonymity. In addition, Prime III records only output from the voting machine, but not inputs from the voter, such as the location where voters touch the screen. Our work extends their approach by recording all I/O experienced by the voter, and by better protecting voter anonymity.

The independent audit framework approach of Garera et al. monitors the behavior of the voting machine by analyzing its video output in real time. They use computer vision techniques to infer important state transitions (such as a candidate being selected or a vote being cast) and then they log these inferred transitions. In comparison, our system does not try to analyze this data in real time, but rather logs all I/O so that this data can be analyzed after the election. They use a virtual machine monitor to isolate the monitoring system from the voting machine. While their approach is good for voter anonymity because it does not record I/O, it does not save as much evidence for post-election investigations.

Ptouch, a predecessor to Pvote, takes a step in the direction of replayable audit logs (86). Ptouch does not record any I/O during the process of voting and does not record full screen images, but it does introduce the idea of recording data exactly as it is seen by the voter. For each candidate selected by the voter, Ptouch records a bitmap image of the candidate's name (as seen and selected by the voter during the voting process) in the electronic cast vote record.

The Auditorium project developed techniques for ensuring the robustness and integrity of event logs using locally networked voting machines (69). Their work studies how to log data, while we examine the question of what to log, so their work is complementary.

Molnar et al. (58) introduced the notion of history independence to vote storage on voting machines.

A review (85) of voting machine firmware used in Sarasota County motivated our study of audit logs that capture user behavior in greater detail than found in current voting systems. In that election, some voters alleged that the CD13 contest was never displayed to them, that their initial selection did not appear on the confirmation screen, or that the voting machine did not recognize their attempts to select one candidate in that contest. Unfortunately, the audit logs available in that election were limited. If voting machines in Sarasota County had been equipped with the kind of audit log mechanism proposed in this chapter, investigators would have had considerably more evidence to investigate these allegations and would have been able to replay voter sessions to see whether these allegations were accurate.

Bellovin first proposed the idea of full-interaction audit logs to us in private communication in 2004 (10). Tyson later independently proposed a similar concept, motivated by his work in the Sarasota County voting review (85). This research explores this concept in greater detail.

3.6 Conclusion

We propose a method for recording reliable audit logs directly that record the interactive behavior of the voting machine *as it is experienced by the voter*. We call these *replayable* audit logs, and we show they can be generated by recording data directly from the I/O of the voting machine. We also show how to protect ballot secrecy by logging frames of video only when the image changes and by protecting the confidentiality of audit logs. As a result, this approach allows useful evidence of voter intent to be logged, while protecting the anonymity of the voter. Our prototype implementation demonstrates the practicality as well as limitations of the approach.

Chapter 4

Efficient User-Guided Ballot Image Verification

Optical scan voting systems are ubiquitous. Unfortunately, optical scan technology is vulnerable to failures that can result in miscounted votes and lost confidence. While manual counts may be able to detect these failures, counting all the ballots by hand is in many situations impractical and prohibitively expensive. In this chapter, we present a novel approach for examining a large set of ballot images to verify that they were properly interpreted by the opscan system. Our system allows the user to *simultaneously* inspect and verify many ballot images at once. In this way, our scheme is significantly more efficient than manually recounting or inspecting ballots one at a time, providing the accuracy associated with human inspection at reduced cost. We evaluate our approach on approximately 30,000 ballots cast in the June 2008 Humboldt County Primary Election and demonstrate that our approach improves the efficiency of human verification of ballot images by an order of magnitude.

4.1 Optical scan systems

In recent years, optical scan systems, where paper ballots are marked by a voter and read by a machine, have seen increasing adoption throughout the U.S. At present, nearly two-thirds of voters vote using optical scan technology (79). Unfortunately, there is sometimes a disconnect between the way a voter (or election official) might interpret a mark and the way a machine interprets it. This disconnect can lead to lost or miscounted votes and even to a loss of trust if these errors become too frequent.

At a high level, optical scan systems work by scanning a paper ballot, determining its layout by reading a machine-readable code on the ballot (e.g., a barcode), and then interpreting the voter-marked regions on the ballot associated with each vote (the *voting targets*) (50). Depending on the ballot layout, a filled voting target may indicate a vote for "Yes" or "No" on a contest or a vote for a particular candidate.

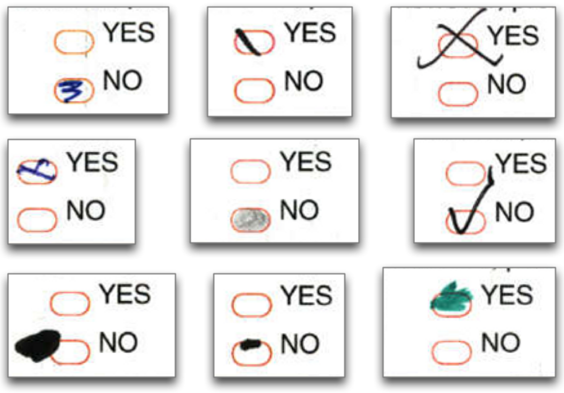


Figure 4.1. Examples of *marginal marks* from our data set according to our classification procedure (see Figure 4.15). Commercial opscan systems generally do not make clear where the boundary lies between marks that will be classified as filled and those that will be interpreted as empty. Marks near that boundary, such as these, may not be reliably (or even consistently) classified the way the voter intended.

A simple and common approach to mechanically detect whether a voting target is marked is to compute the average pixel intensity value of the area inside the voting target, classifying it as marked if the average falls beyond a predetermined threshold. Typically, the vast majority of voting targets fall into one of two cleanly separated distributions, according to whether the voting target was marked or not (as in Figure 4.15). However, the threshold that optical scanners and their software use to determine whether a voting target is filled or empty can be somewhat arbitrary and is unknown to the end user. In addition, not all marks fall cleanly into the "filled" and "empty" classes. In the sample of 60,000 voting target images that we analyze in this research, 150 (0.25% of the total) are *marginal*, in that their average intensity is neither near the typical value for a filled target nor near the typical value for an empty target (for examples, see Figure 4.1). Current opscan systems often have trouble interpreting these marginal marks accurately.

Another serious problem is that the classification of voting targets as filled or empty is not a perfect indication of the voter's intent. For example, a voter, perhaps unaware of the mechanical process used to interpret his or her ballot, may have marked the ballot reasonably, but in a way that is misinterpreted by the scanner. We call these *mark-sensing failures*. See Figure 4.2 for two examples of mark-sensing failures. Mark-sensing failures have impacted elections in the past (3; 30).

A second category of optical scan system failures is *configuration failures*. Beyond sensing marks, an optical scan system must determine which candidate or contest choice each mark corresponds to, and whether the marks should be counted as valid votes. For example, in a vote-for-one contest, marking two selections should count as vote for neither (i.e., an *overvote*). The logic regulating these decisions can get complex, leading to inadvertent errors. Errors in the opscan configuration can cause votes to be attributed to the wrong candidate, cause valid votes to be mistakenly treated as overvotes, or cause overvotes to be

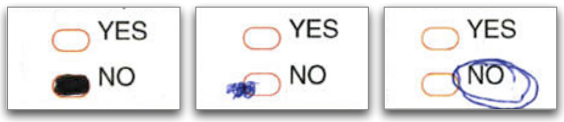


Figure 4.2. Three real-life examples of “No” votes: a correctly classified filled voting target (left), a marginal mark (center), and a mark that was incorrectly classified as an empty voting target because there is little ink inside the voting target itself (right). The latter two cases are examples of potential mark-sensing failures of an optical scan system.

mistakenly interpreted as valid votes. Configuration failures have impacted elections in the past (67; 89; 14).

These kinds of failures have the potential to change election outcomes, so it is important to have some way to detect and correct them. For example, the 2008 Minnesota Senate race was initially decided by a margin of 206 votes out of 2.9 million optical scan ballots cast (62; 35). After a lengthy recount of all the ballots and challenges over voter intent, the outcome of the election was reversed.

As in Minnesota, one approach to verifying an election is to count the paper ballots manually in either a full recount or a statistical random audit (38). Unfortunately, full recounts are expensive and rare, and random audits do not detect all instances of mark-sensing and configuration failures.

Another approach (pioneered by the Humboldt County Election Transparency Project (42)) is to independently rescan the paper ballots with an off-the-shelf document scanner and publish the digitized images. While now the digitized images must be trusted, this allows other interested parties to verify the classification of these images into votes, either manually, or by writing their own software to do so.

Publication of ballot images enables an independent automated recount or manual analysis of the ballots. The hope is that two independent systems will fail independently, thus reducing the risk of undetected errors. However, this hope is not always borne out in practice, due to the risk of common-mode failures and correlated errors. For example, two systems that both interpret voting targets as filled or empty based on the image intensity inside the voting target may both fail to correctly identify the voter’s intent when the voter has expressed his or her intent in an unanticipated way (see Figure 4.2).

In this research, we focus on trying to confirm whether the opscan system has accurately interpreted a set of ballot images. We define accuracy in terms of how a human examining those images would interpret the voter’s intent on each ballot. The difference between human interpretation and machine interpretation can be significant, particularly when interpreting marginal or ambiguous marks. Unfortunately, while human interpretation can be very effective at ascertaining voter intent, it is also time-consuming and prohibitively expensive to perform at large scale. Reducing the cost of human interpretation is the goal of this work.

4.1.1 Problem statement

Given a set of ballot images of an election and the opscan system's interpretation of each ballot, we want to allow the user of our system to efficiently check the machine's interpretations. We want to detect every error that would have been found through individual inspection of every ballot image, but we want to do this more efficiently. In other words, our goal is to make it easier for a user to confirm that all ballot images were interpreted accurately, verifying for him or herself that all votes have been counted correctly and that all anomalous ballots are accounted for. The user should not have to take it on faith that the optical scanner interpreted the ballots correctly: the user should be able to detect any and all ballots that may have been interpreted inaccurately, whether due to buggy code or to imperfect classification algorithms.

Currently, the most rigorous way to establish that a set of ballot images has been counted accurately is to count them one by one. However, manual recounts are time-consuming and difficult. It is possible to develop software to analyze and automatically classify votes, flagging anomalous or marginal votes for later inspection. However, the accuracy of the software's interpretation of the ballots and the effectiveness of the software's algorithms must be trusted by the user. In addition, such software is also subject to the same mark-sensing and configuration failures as optical scanners, because everything is automated. We seek some way for the voter to confirm that the software interpreted the ballots accurately. *It is hard to establish the accuracy of the count without actually* ***inspecting*** *the ballots.*

We make a few assumptions for the purposes of this work. First, we assume that the given ballot images are authentic, and that they accurately represent the complete set of ballots to be investigated. Verifying this is an orthogonal issue best addressed through other mechanisms (e.g., random auditing and chain-of-custody protections). Second, we assume our user's subjective interpretation of a ballot is the ground truth. Our goal is simply to present the data efficiently for the user to verify. We make no attempt to determine how other users might interpret the ballot images, and we do not claim that our techniques will resolve disputes in cases where different people interpret the same ballot differently.[1] Third, we make no attempt to defend against malicious code, malicious insiders, or malicious attacks on the voting system. Our focus is on detecting common errors that may occur despite the best intentions of all parties.

Contributions of this research. We propose a method of visualizing and checking for errors in the interpretation of a collection of ballot images, by applying simple operators to the images and displaying them in superimposed form. We develop an interactive verification process that significantly improves efficiency and reduces the burden on the user of the system while at the same time allowing individual treatment of each anomalous ballot. We demonstrate the effectiveness of our approach by using it to obtain a human-verified interpretation of 30,000 votes cast in Proposition 98 in Humboldt County, California. We

[1]It is not unusual for two people to differ in their interpretation of voter intent when the stakes are high, as is evident in the 2008 Minnesota Senate election recount (62).

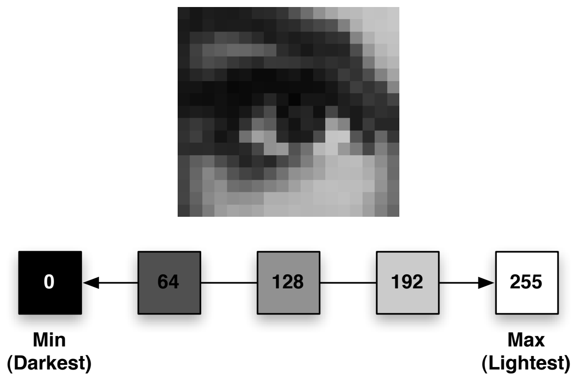


Figure 4.3. An image represented by a matrix of pixels (top). The intensity at each pixel is represented by an 8-bit value ranging from 0 to 255 (bottom). The minimum of a set of pixel values is therefore the darkest pixel, while the maximum is the lightest.

use this interpretation to identify a number of anomalies and unusual marks found in these ballots and quantify their prevalence.

4.2 Superimposing ballot images

The primary contribution of this research is a method of verifying a large set of ballot images by inspecting a small number of superimposed images. In this section we introduce the idea and theory of two types of superimposed images, *min-images* and *max-images*, and how they apply to verification. To introduce this concept we briefly review the basics of digital images.

A digital image (e.g., a scanned paper ballot) may be thought of as a matrix of *pixels*, each corresponding to the color or intensity of the image at that position. In 8-bit grayscale images, pixel values are numbers between 0 and 255 that represent the brightness, or *luminance intensity*, of the pixel (see Figure 4.3). Color images are similar, except that they are usually composed of multiple *channels* (e.g., red, green, and blue channels), instead of just one (luminance).

4.2.1 Min-images and max-images

Consider a set of N grayscale images of the same contest, each $I \times J$ pixels in size and aligned to one another (or *registered*). Two interesting operations we can consider on these images are the minimum and maximum operations. As shown in Figure 4.4, for each pixel position (i, j), we compute the minimum and maximum at that position in each of the N images. The result of these operations are stored in the *min-image* and *max-image*, respectively.

If all the images are identical, the value of the min- and max-images would also be identical. However, variation in any pixel of the N images will result in differing minimum

```
MIN-AND-MAX-IMAGES($images$)
1  $min\_image \leftarrow$ new solid *white* image of size $I \times J$
2  $max\_image \leftarrow$ new solid *black* image of size $I \times J$
3  **for** $j \leftarrow 0$ **to** $J - 1$
4  **do for** $i \leftarrow 0$ **to** $I - 1$
5      **do for** $image$ **in** $images$
6          **do** $min\_image_{i,j} \leftarrow min(image_{i,j}, min\_image_{i,j})$
7             $max\_image_{i,j} \leftarrow max(image_{i,j}, max\_image_{i,j})$
8  **return** $(min\_image, max\_image)$
```

Figure 4.4. Pseudocode for calculating min- and max-images.

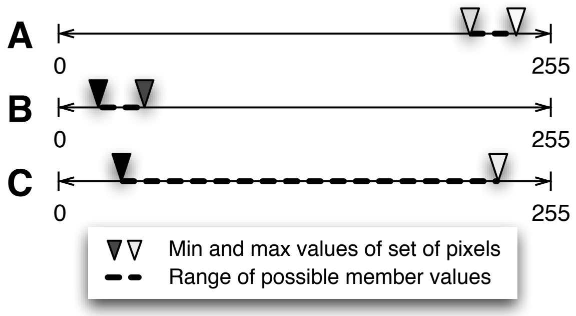

A
0
255
B
0
255
C
0
255
▼▽ Min and max values of set of pixels
Range of possible member values

Figure 4.5. Three example cases, representing the minimum and maximum intensity values of a set of images at three pixel locations, A, B and C. In cases A and B, the min- and max-images have almost the same value at that pixel location, indicating that there is little variation among the pixel values at that location from image to image. Case C gives us fewer guarantees about the member values; all we know is that there is broad variation between at least one pair of images at that pixel location. For instance, this variation could be due to a stray mark at that pixel location in one ballot image, which might warrant further investigation.

and maximum images, and the difference between the minimum and maximum at any pixel location defines a range in which all pixels in the set necessarily lie (see Figure 4.5).

We can think of how the minimum and maximum operations would appear visually for a given pixel location: because pixel values correspond to luminance intensity, the smaller the value, the darker the pixel becomes. Therefore, the minimum operator on a set of pixels is equivalent to taking the darkest pixel in the set. For example, if even just one pixel is black, the resulting minimum pixel will be black. Similarly, the maximum pixel value corresponds to the lightest among a set of pixels. If any pixel in the set is white, the maximum will also be white.

4.2.2 Using superimposed ballot images for verification

We leverage the min- and max-images for simultaneous verification of large sets of aligned ballot images.

The min-image. If we make the assumption that when a voter marks a blank ballot, he or she *darkens* the image, we can consider the min-image to consist of the *union* of all of the voters' marks, superimposed on a single image. In other words, if one ballot image contains an extraneous mark, the min-image will also contain that mark, as well as any other marks that exist in the set of ballot images.

If we see *no* questionable stray marks in the min-image, we are guaranteed that *none* of the ballot images contain stray marks. If we do see one or more questionable marks, we know that there exists at least one stray mark in the ballot images and we must investigate them further.

Beware: a min-image that looks like a correctly marked ballot (e.g., a valid "Yes" vote) does not guarantee that all (or any) of the ballots in the set are correctly marked. The min-image can be used to verify the *absence* of any marks at a given location across all of the ballots, but it cannot be used to verify the presence of a mark in all ballot images.

The max-image. While the min-image can be considered the union of all voter marks, the max-image can be thought of as the *intersection*. In other words, if any ballot is white at a particular location, then the max-image will also be white at that location. This property can be used to detect whether the collection contains a ballot with whitespace where it should not be, such as an incompletely filled voting target or missing text that was supposed to be printed on the ballot but is not present in the scanned image. Therefore, if we see a valid mark in the max-image, we are guaranteed that *all* ballots in the set also contain that mark.

Verifying ballot images. The use of min-images and max-images can thus let us establish strong guarantees about the superimposed ballots:

- If we see a pair of min- and max-images that are similar (or identical) we know that all ballots contained are similar. For example, see Figure 4.6.
- If we see a pair of min- and max-images that are dissimilar (e.g., because of a stray mark or under-filled voting target) then we cannot verify the ballot images, and further investigation is required. For example, see Figure 4.7.

The min/max operations can be applied to any number of images. For example, take the two images shown in Figure 4.7. From the image on the left we learn that at least one of the 800 ballots contains an anomalous arrow pointing to the "Yes" label. Similarly we see that at least one ballot appears to contain the initials "CC" We also see that at least one ballot contains a hesitation mark in the "No" voting target.

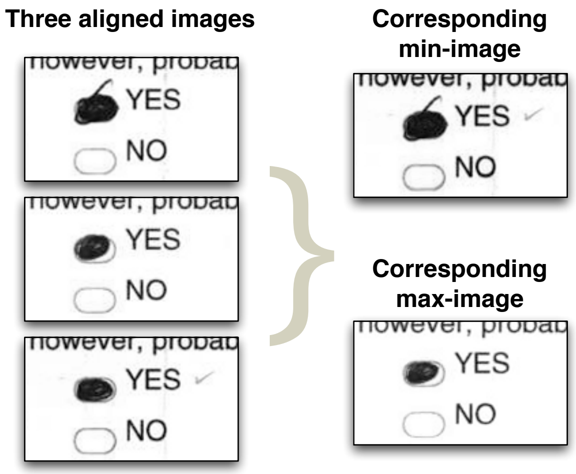


Figure 4.6. A visual example of the min and max image operators. From the min-image we notice a subtle check mark, indicating the existence of such a stray mark in one of the superimposed ballots. We also notice that none of the superimposed images have any marks whatsoever that could be construed as a "No" vote. The max-image shows us what the ballots have in common: that they all have a significant amount of ink in the "Yes" voting target. From the two superimposed images, we can reasonably conclude that all three ballots are unambiguous "Yes" votes.

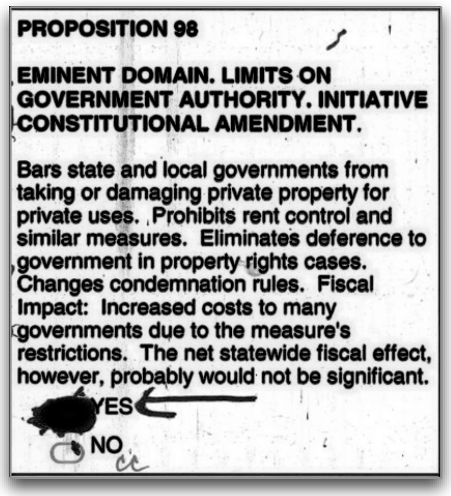
From the image on the right we learn that all 800 ballots contain at least as much shading in the "Yes" voting target as is shown in that image. We also learn that all 800 ballots correctly contain the same textual content; no ballot in the set omits or alters any text.

From these two images, barring the extraneous marks that we may want to investigate, we can reasonably conclude that all votes in the stack should be classified as "Yes" votes. Through the use of these two images we are able to establish bounds on the amount of variation that exists in the 800 images. This enables a user to efficiently verify a large set of ballot images simultaneously.

4.2.3 Overlay images

One slightly unwieldy aspect of examining min- and max-images is that the operator must examine two images at the same time. However, it is possible to improve the presentation by summarizing all the information in a single image: the *overlay image*. The key observation is that the max-image is a "subset" of the min-image: every pixel in the min-image is at least as dark as the corresponding pixel in the max-image. Because we have elected to work with grayscale images, it is possible to combine the min- and max-images into a single image by assigning one color to the max-image and another color to the min-image and superimposing the two (see Figure 4.8). We call the superimposed false-color image an *overlay image*. Example overlay images are shown in Figure 4.9.

800 ballots superimposed with *min* (darkening) operator


PROPOSITION 98

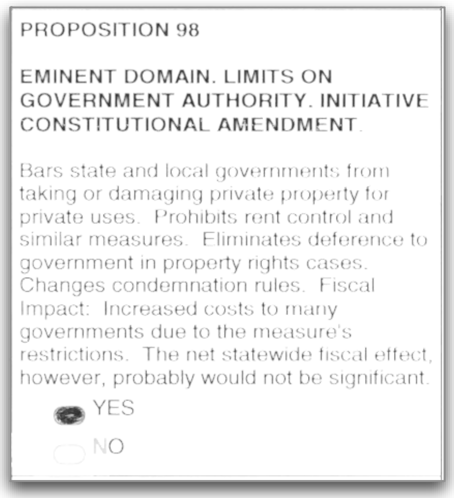
EMINENT DOMAIN. LIMITS ON GOVERNMENT AUTHORITY. INITIATIVE CONSTITUTIONAL AMENDMENT.

Bars state and local governments from taking or damaging private property for private uses. Prohibits rent control and similar measures. Eliminates deference to government in property rights cases. Changes condemnation rules. Fiscal Impact: Increased costs to many governments due to the measure's restrictions. The net statewide fiscal effect, however, probably would not be significant.

[✓ YES]

NO

800 ballots superimposed with *max* (lightening) operator


PROPOSITION 98

EMINENT DOMAIN. LIMITS ON GOVERNMENT AUTHORITY. INITIATIVE CONSTITUTIONAL AMENDMENT.

Bars state and local governments from taking or damaging private property for private uses. Prohibits rent control and similar measures. Eliminates deference to government in property rights cases. Changes condemnation rules. Fiscal Impact: Increased costs to many governments due to the measure's restrictions. The net statewide fiscal effect, however, probably would not be significant.

[✓ YES]

NO

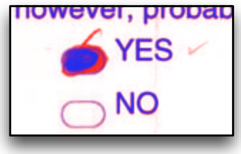
Figure 4.7. A superposition of 800 ballots, all initially classified as "Yes" votes. The min-image is shown on the left, and the max-image on the right. From these two images we conclude that further investigation is required to identify the ballots with stray marks. However, we also learn from the left image that *none* of the 800 voters (completely) filled the "No" choice, and we learn from the right image that *all* 800 voters filled the "Yes" choice at least that much, so we can reasonably predict that the superimposed ballots are all "Yes" votes.

```
OVERLAY-IMAGE($min\_image, max\_image, min\_color, max\_color$)
1  $overlay\_image \leftarrow$ new $I \times J$ color image
2  for $j \leftarrow 0$ to $J - 1$
3  do for $i \leftarrow 0$ to $I - 1$
4      do $\alpha_{min} \leftarrow min\_image_{i,j} / 255$
5         $\alpha_{max} \leftarrow max\_image_{i,j} / 255$
6         $t \leftarrow (1 - \alpha_{min}) \cdot min\_color + \alpha_{min} \cdot white$
7         $overlay\_image_{i,j} \leftarrow (1 - \alpha_{max}) \cdot max\_color + \alpha_{max} \cdot t$
8  return $overlay\_image$
```

Figure 4.8. Pseudocode for creating a single overlay image from a given min- and max-image. An overlay image is created by interpolating or *alpha blending* first between the background color (white) and the *min-color*, and then between the resulting color and the *max-color*, in other words treating the min- and max-images as *alpha layers*. In Section 4.5.4 we describe and compare this to an alternate overlay image algorithm.

Examples of overlay images with
***min* in red and *max* in blue**

3 ballot images

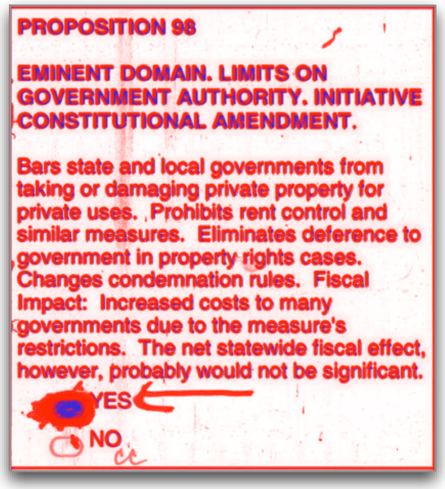


however, probab

YES

NO

800 ballot images



PROPOSITION 98

EMINENT DOMAIN. LIMITS ON GOVERNMENT AUTHORITY. INITIATIVE CONSTITUTIONAL AMENDMENT.

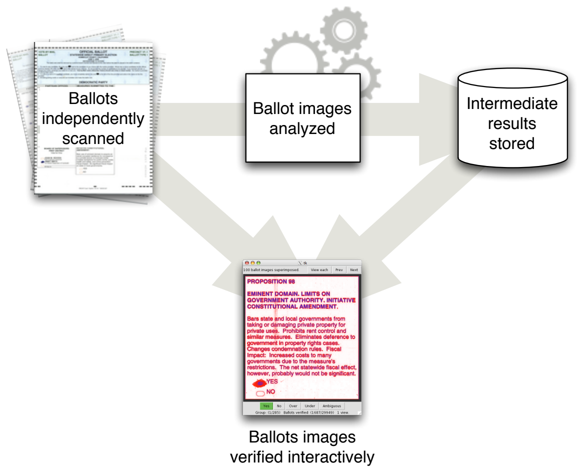
Bars state and local governments from taking or damaging private property for private uses. Prohibits rent control and similar measures. Eliminates deference to government in property rights cases. Changes condemnation rules. Fiscal Impact: Increased costs to many governments due to the measure's restrictions. The net statewide fiscal effect, however, probably would not be significant.

YES

NO

Figure 4.9. Three images from Figure 4.6 represented by a single overlay image (top). The result of superimposing the min-image (as red) and max-image (as blue) from Figure 4.7, to obtain an overlay image (bottom).

The examples in this chapter use red for the min-image and blue for the max-image.[2] Therefore, blue regions of the overlay image correspond to locations where every ballot is equally dark (e.g., due to marks or printing present on every ballot in the set), and red regions correspond to locations where at least one ballot is dark and at least one is light (e.g., due to stray marks or variation from ballot to ballot). Text printed on the blank ballot appears in blue on the overlay image. The surrounding red "halo" is due to imperfect registration and alignment of images.


Ballots independently scanned
Ballot images analyzed
Intermediate results stored
tk
View each
Prev
Next
Yes
No
Over
Under
Ambiguous
Ballots images verified interactively

100 ballot images superimposed.

PROPOSITION 98

EMINENT DOMAIN. LIMITS ON GOVERNMENT AUTHORITY. INITIATIVE CONSTITUTIONAL AMENDMENT.

Bars state and local governments from taking or damaging private property for private uses. Prohibits rent control and similar measures. Eliminates deference to government in property rights cases. Changes condemnation rules. Fiscal Impact: Increased costs to many governments due to the measure's restrictions. The net statewide fiscal effect, however, probably would not be significant.

YES

NO

Group: (1/285) Ballots verified: (1687/29949) 1 view.

Figure 4.10. The proposed workflow for our system. Ballots are first scanned and analyzed. The ballot images are tentatively classified as votes for a particular choice or candidate (see Section 4.4.2). The tentative classifications are then verified using our interactive image-overlay tool (see Section 4.3).

4.3 Verifying elections using overlay images

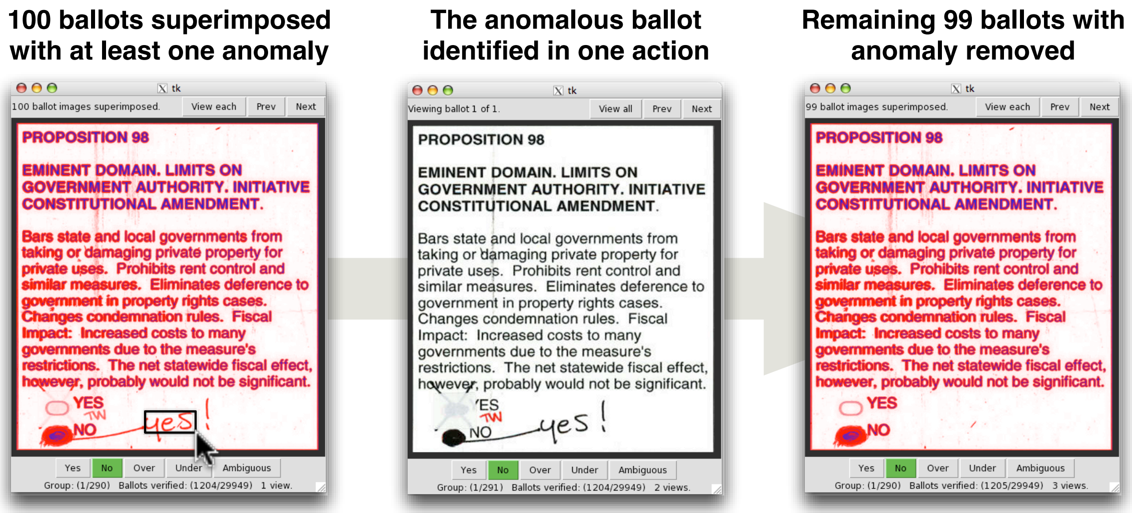
We envision that overlay images could form part of a system for auditing and verifying the interpretation of marked ballots. The process, illustrated in Figure 4.10, can be split into three parts, each with a different set of considerations:

1. *Image scanning*: This is the process of taking paper ballots and scanning them into an image format for use by our system. Several factors here may influence accuracy later, such as skew angle, image resolution, scanner noise, and artifacts from image compression. The details of how images are scanned is beyond the scope of this research, and we consider the scanned images as inputs into our system.

2. *Image analysis*: This is the process of grouping and aligning the ballots, identifying and segmenting the regions for each contest of interest, and tentatively classifying the marks of the voter. Note that automatic classification of voter marks is not the primary focus of this research; our goal is to *verify* the accuracy of such a subsystem. We could have, for instance, replaced the classification procedure with another software component or with the official optical scan system.

3. *Image verification*: In the image verification stage, the user (e.g., a voting official certified to make judgment calls) checks the accuracy of the interpretations made in the image analysis stage. Our system provides an interactive tool that enables the user to review, verify, and if necessary correct the interpretation of ballot images as votes. We implement a simple user interface to let the user efficiently validate large batches

[2]Note that red and blue are parameters to our overlay image approach; any two colors may be used instead to account for the perceptual needs or preferences of the user.


100 ballots superimposed with at least one anomaly
The anomalous ballot identified in one action
Remaining 99 ballots with anomaly removed
tk
View each
Prev
Next
View all
Yes
No
Over
Under
Ambiguous
yes!

100 ballot images superimposed.

PROPOSITION 98

EMINENT DOMAIN. LIMITS ON GOVERNMENT AUTHORITY. INITIATIVE CONSTITUTIONAL AMENDMENT.

Bars state and local governments from taking or damaging private property for private uses. Prohibits rent control and similar measures. Eliminates deference to government in property rights cases. Changes condemnation rules. Fiscal Impact: Increased costs to many governments due to the measure's restrictions. The net statewide fiscal effect, however, probably would not be significant.

YES

NO

Group: (1/290) Ballots verified: (1204/29949) 1 view.

Viewing ballot 1 of 1.

PROPOSITION 98

EMINENT DOMAIN. LIMITS ON GOVERNMENT AUTHORITY. INITIATIVE CONSTITUTIONAL AMENDMENT.

Bars state and local governments from taking or damaging private property for private uses. Prohibits rent control and similar measures. Eliminates deference to government in property rights cases. Changes condemnation rules. Fiscal Impact: Increased costs to many governments due to the measure's restrictions. The net statewide fiscal effect, however, probably would not be significant.

YES

NO

Group: (1/291) Ballots verified: (1204/29949) 2 views.

99 ballot images superimposed.

PROPOSITION 98

EMINENT DOMAIN. LIMITS ON GOVERNMENT AUTHORITY. INITIATIVE CONSTITUTIONAL AMENDMENT.

Bars state and local governments from taking or damaging private property for private uses. Prohibits rent control and similar measures. Eliminates deference to government in property rights cases. Changes condemnation rules. Fiscal Impact: Increased costs to many governments due to the measure's restrictions. The net statewide fiscal effect, however, probably would not be significant.

YES

NO

Group: (1/290) Ballots verified: (1205/29949) 3 views.

Figure 4.11. Our user interface showing how an anomaly can be easily identified and separately verified. 1.) An overlay of 100 "No" ballots presented; the user notices an anomaly (a written "Yes!") and selects it by drawing a box (left). 2.) Our system finds the one ballot containing the highlighted anomaly and displays it for the user to verify separately (center). 3.) The user verifies the remaining 99 ballots, now clean of any questionable marks (right).

of ballots and hone in on problem or anomalous ballots. The image verification stage is the core of our contribution, and we expand below on the major elements of this approach to image verification.

4.3.1 Grouping ballots into verifiable sets

Given a set of aligned and classified ballot images, we turn our attention to the process of verifying the classification of each image as a set of votes.

Before we can verify ballots using our min/max overlay approach, we must group ballots into visually similar sets. For instance, we group all the "Yes" votes, "No" votes, overvotes, and undervotes together. We will verify each of these four groups separately. Note, each group may have many images in it. In our experiment described below there were over 10,000 ballots in each of the "Yes" and "No" groups.

If the ballot images are very well aligned, and if there is very little noise in the images, it may be possible to overlay all the ballots in one group simultaneously. However, in practice, there is a limit to the number of images that can be superimposed before the noise becomes too great. Thus, it is helpful to break down each group into smaller groups (e.g., of maximum size 200). This still lets us verify many ballots at a time, while limiting the effect of noise.

4.3.2 User interface to verify ballot groups

Once we have grouped ballot images into visually similar groups, our tool allows the user to step through them, one group at a time, to verify them. To do so we created a graphical user interface that sequentially presents the user with the min/max overlay image for each group.

The verification tool begins with all the ballot groups in a queue, presenting the overlay image for each group one at a time. If the overlay image looks correct to the user, he or she can verify the group of ballots with one click. When a group is verified, it is removed from the queue. When the queue is empty, all ballots have been verified. The interesting part of the process happens when an anomaly is detected, or when a group cannot immediately be verified.

4.3.3 Dealing with anomalies

In Figure 4.11 we see an example of the user interface displaying the overlay image for a group with an anomaly in it. We see that at least one ballot has a big "Yes!" written on it. In order to verify this group we need to deal with this anomalous ballot.

We approach this problem by letting the user easily partition groups of ballots based on a region of interest. We allow the user to select a rectangular region in the overlay image, and then we split the group of ballot images into two subgroups based upon their contents within this rectangular region. In the example shown in Figure 4.11, when the user draws a rectangle around the "Yes!", the initial group of 100 ballots is split into two groups (in this case, one group of one ballot, shown center, and one group of 99 ballots, shown right). This can be effectively accomplished by examining the average intensity for each ballot in the region of interest. A standard clustering algorithm can be used to partition the group. We used K-means (54), with the number of clusters equal to two. The algorithm for partitioning ballot groups is given in Figure 4.12. The original group becomes two disjoint groups in the queue of unverified ballots.

This approach is quite general. The user, for example, is not limited to selecting regions with stray marks. Because K-means simply clusters images by similarity in average image intensity at a region, the user may select any region as a basis for clustering. This approach allows a user to interactively partition a large set of images, if necessary, into smaller groups that can then be verified. It is important to note that the quality of the initial ballot groupings affects only the *efficiency* of verification—*not the correctness*—because the user may repartition groups into smaller and smaller groups (individual ballots, if necessary) until he or she is satisfied with the verifiability of the resulting groups.

```
PARTITION-IMAGES($images, region$)
1  $sum\_intensities \leftarrow$ new ASSOCIATIVE-ARRAY
2  for $image$ in $images$
3  do $sum\_intensities[image] \leftarrow 0$
4      for $j \leftarrow region.y$ to $(region.y + region.height - 1)$
5      do for $i \leftarrow region.x$ to $(region.x + region.width - 1)$
6          do $sum\_intensities[image] \leftarrow sum\_intensities[image] + image_{i,j}$
7  $(cluster_1, cluster_2) \leftarrow$ K-MEANS($sum\_intensities, k = 2$)
8  return $(cluster_1, cluster_2)$
```

Figure 4.12. Pseudocode for the approach we use to find a group of anomalous ballots, given a set of images and a region of interest. In this code, K-MEANS refers to the standard clustering algorithm that, in this case, partitions a set of scalar values into two clusters centered around two group means in a way that best explains the data (54).

4.4 Evaluation

In this section we discuss an experiment we ran to evaluate the practicality of our approach.

4.4.1 Dataset

We evaluate our system using ballot image data from the June 2008 Humboldt County California Primary Election. The ballot scans are JPEG-compressed color images at a resolution of 150 DPI, which we obtained from the Humboldt County Election Transparency Project (42). They include scans of both mail-in and precinct-cast ballots for a Diebold/Premier optical scan system. As this was a primary election, they include ballots for multiple parties. We limit our scope to evaluating one statewide contest, Proposition 98, that appears in each of the 29,949 ballot images in our data set.

4.4.2 Image analysis

The goal of the image analysis stage is to register the images to the same coordinate system, locate the contest of interest on each image (Proposition 98 in our case), and classify the votes as *Yes, No, undervote,* or *overvote* (see Figure 4.13). These classifications will be verified in the verification stage.

Image registration. For our overlay approach to work we require precise alignment (or *registration*) of the images to one another. To register the images, we first select features we expect to appear in the same place on all our ballots. The features we used in our experiment were the inside corners of the four outermost hash marks (as depicted in Figure 4.14).

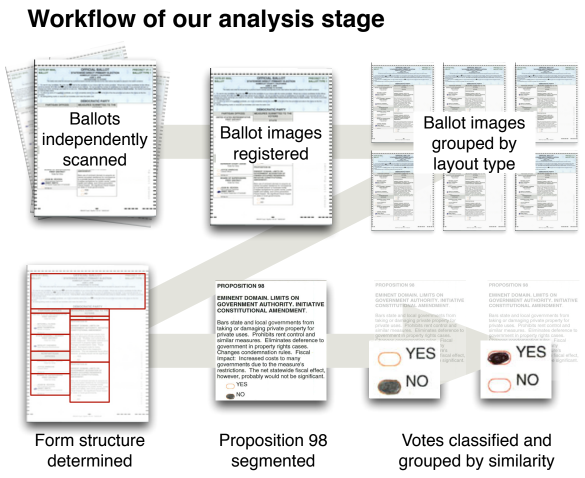

Workflow of our analysis stage
Ballots independently scanned
Ballot images registered
Ballot images grouped by layout type
Form structure determined
Proposition 98 segmented
Votes classified and grouped by similarity
YES
NO

Figure 4.13. An illustration of the techniques we use for analyzing ballot images and interpreting the marks on these images. The images are registered (mapped to a universal coordinate system and aligned), classified by ballot layout type, segmented into regions for each contest and voting position, classified as votes for candidates, and finally broken down into smaller groups, optionally by image similarity.

Next, for each ballot image, we locate the selected features and compute an affine transformation that best maps these points to the four corners of the common coordinate space. We then apply this transformation to the ballot image, which accomplishes the necessary rotation, translation, and scaling necessary to align all the images. Note that an incorrect transformation can be detected in the verification stage.[3]

We took the approach of registering the entire ballot, and then cropping it to retain the region of interest (i.e., Proposition 98 in our example). In hindsight, it may have been beneficial to re-register the images after cropping, because small errors introduced in locating the various regions accumulate when the cropped images are overlaid.

Form extraction. After registering the images, we determine the structure of the ballots through *form extraction* (20), and use this to locate the coordinates of every box in the image. In our case study, we used a heuristic to identify the location of Proposition 98 on each ballot. We made an assumption (subsequently verified using our approach) that the size and shape of the contest box was constant, no matter where it appeared on the ballot.[4] For Proposition 98, we use the size of the contest box to locate it on the page, but it may be useful to examine other information as well, such as voting target positions, OCR'd text, or

[3]We also found we could reliably detect mis-registered ballots before the verification stage, by examining the *residual* of the affine transformation; the residual was high whenever our simple feature detection algorithm failed to locate the correct features, typically due to the existence of stray marks. This affected 26 of the $29,949$ ballots, which we then manually registered.

[4]In our case study, Proposition 98 appeared in exactly six locations on the ballot, despite there being many more ballot types (784 as counted by unique barcodes); experimentally, the box dimensions for Proposition 98 varied from ballot to ballot by 1–2 pixels.

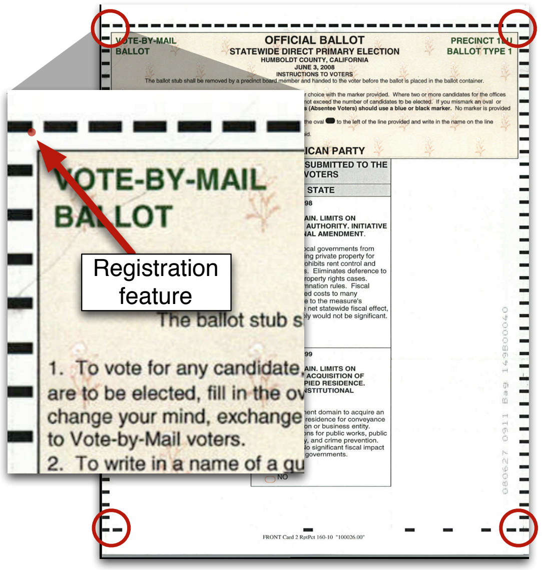


Figure 4.14. To precisely register the ballot images in our set, we first locate the innermost corners of the four outermost hash marks. Given the location of these four points we compute an affine transformation that best maps these four points to corresponding locations in a known coordinate system.

other pattern-matching techniques. Finally, given the coordinates of the contest of interest, we crop the image and identify the voting targets.

Initial classification. To classify the ballot images, we calculated the average pixel intensity within each voting target region, and applied a threshold to classify that voting target as filled or empty. A histogram of the average pixel intensities is shown in Figure 4.15. The threshold we used in our experiments was the halfway-point between labels B and C.

We initially categorized each Proposition 98 contest into a predicted category: *yes*, *no*, *undervote*, or *overvote*. The classification here is not meant to be final, as it will be verified later. We use the classifications to create overlay images of similarly voted ballots for the user to more efficiently verify.

Grouping ballots into verifiable sets. Within each of the four categories, we further grouped the ballots into groups of 200 randomly selected images. The number 200 here was heuristically chosen to balance efficiency and minimize the effect of noise. We did not attempt to optimize the group size for this work. We expect we could gain more efficiency by grouping ballots in a smarter way, for instance by clustering images based on visual similarity, as we discuss later.

After grouping the images into smaller, visually similar groups, our system has all the information it needs to present the results to the user for interactive verification using our overlay image method. In the next section, we evaluate this approach.

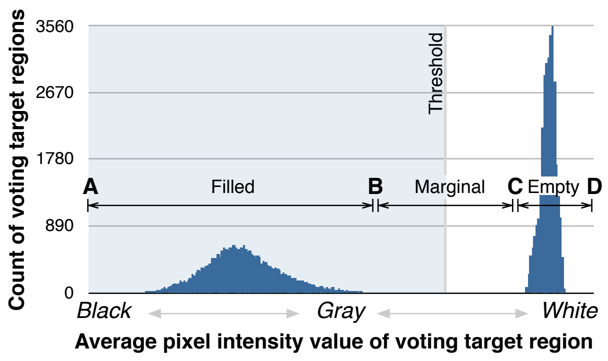


Figure 4.15. Histogram of average pixel intensity values of 59,898 voting target images taken from Proposition 98 of the June 2008 Primary Election in Humboldt County, California. Despite the apparent separation between modes, 150 (or 0.25%) voting target images fall between labels B and C.

4.4.3 Evaluating verification efficiency

We ran two experiments. First, we counted all ballots using our proposed method and user interface. Second, we compared this to counting the ballots individually. For the latter case, for the sake of time, we visually inspected 1,000 of the 29,949 ballots. We extrapolate from this an estimate of the total time it would take to individually inspect all ballots.

Results counting one by one. To have something to compare our results to, one of the authors examined 1,000 ballot images, one by one. This took 23 minutes. Extrapolating that rate to the full data set means we could have verified the entire set in 11 or 12 hours.

Results using our system. Using our system, the same author visually inspected *all* the Proposition 98 ballot images. We inspected the ballot images in groups of at most 200 images at a time, superimposed using the min/max overlay method described in Section 4.2.3. For every anomaly we discovered, we used the group partition method from Section 4.3.3 to identify the anomalous ballots and interpret them individually.

We visually inspected all the Proposition 98 ballot images in *1 hour and 23 minutes.* This represents over an 8× improvement in the time it would take to inspect ballots one at a time. We examined 1,326 images in total, representing a 22× improvement over examining the 29,949 ballot images individually. The results of our verification are shown in Figure 4.16. We acknowledge these are preliminary results—we did not conduct a comprehensive user study—but we argue this is good evidence that our method indeed improves human verification time by an order of magnitude.

All ballot images in data set

After verification

Before verification

	Yes	No	Under	Over	Ambiguous	TOTAL
Yes	11639	0	0	0	1	11640
No	0	16890	0	0	6	16896
Under	15	18	1375	0	0	1408
Over	2	0	0	3	0	5
TOTAL	11656	16908	1375	3	7	29949

All ballot images excluding 150 with marginal marks

After verification

Before verification

	Yes	No	Under	Over	Ambiguous	TOTAL
Yes	11579	0	0	0	1	11580
No	0	16824	0	0	6	16830
Under	0	11	1375	0	0	1386
Over	0	0	0	3	0	3
TOTAL	11579	16835	1375	3	7	29799

The 150 ballot images with marginal marks

After verification

Before verification

	Yes	No	Under	Over	Ambiguous	TOTAL
Yes	60	0	0	0	0	60
No	0	66	0	0	0	66
Under	15	7	0	0	0	22
Over	2	0	0	0	0	2
TOTAL	77	73	0	0	0	150

Figure 4.16. The results of our verification of the Humboldt County Proposition 98 ballot images. The initial tallies from our image analysis stage are compared with the tallies after human verification. The results are further broken down by whether they include marginal marks, as described in Section 4.1. By verifying all votes visually we are able to more accurately interpret some votes that would otherwise have been miscounted.

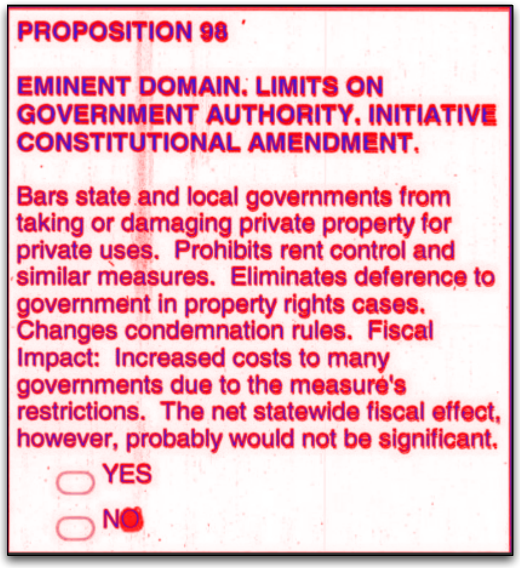
4.4.4 Observations

Using our approach, we were able to verify all steps of the image analysis stage: the image registration, the form extraction, and the vote classification. Had a step failed for any one of the ballots, an anomaly would have been evident in its overlay image. In addition, we detected several interesting patterns of anomalies in the ballots.

One curious pattern was a tendency for a number of voters to fill the incorrect oval when voting for the “No” choice. In particular, they seem to have confused the “o” in “No” for the voting target. These ballots were classified as undervotes by our initial classification, and most likely also interpreted as undervotes in the official canvass. Figure 4.17 shows an example of how this appeared to us in our user interface. This was not an isolated case. Figure 4.18 shows examples of ten voters who consistently mistook the “o” for the voting target.

Another pattern of anomalies we noticed was the recurring tendency to make corrections by drawing arrows to a voting target then annotating it with either “Yes” or “No” (see Figure 4.19). The voter’s intent in some of these cases is unclear.

Last we also noticed cases where extraneous marks appear to be identifying. In Fig-


PROPOSITION 98

EMINENT DOMAIN. LIMITS ON GOVERNMENT AUTHORITY. INITIATIVE CONSTITUTIONAL AMENDMENT.

Bars state and local governments from taking or damaging private property for private uses. Prohibits rent control and similar measures. Eliminates deference to government in property rights cases. Changes condemnation rules. Fiscal Impact: Increased costs to many governments due to the measure's restrictions. The net statewide fiscal effect, however, probably would not be significant.

YES

NO

Figure 4.17. An overlay image that shows at least one anomaly in the “o” in “No.”

ure 4.20 we show two cases where the voter appears to have initialed their changes, which arguably should invalidate their ballot.

4.5 Discussion

In this section we discuss some of the results and consequences of our approach to verification.

4.5.1 Trust in ballot images

We acknowledge that our approach still requires trust in the ballot images, and in the software displaying them (65). Other techniques must be used to gain confidence that the set of ballot images accurately reflects the paper ballots as they were marked by voters. Our approach is designed to minimize the complexity of the software required for verification, and is designed to be intuitive enough for a user to comprehend what is going on. In particular, the minimization and maximization operators were chosen to be simple operations for a machine, as well as simple for a person to reason about. While the software in the image analysis stage—the stage that determines the structure of the ballot images and performs the initial vote classifications—may be complicated, the software necessary to verify a set of images is separable, and can be kept simple. In this way, our system strives to be transparent, a necessity when approaching the problem of verifying elections.

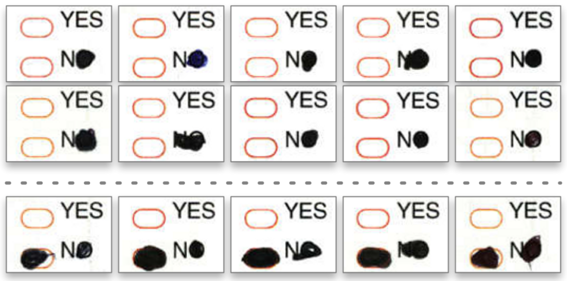


Figure 4.18. A pattern of anomalies we discovered while visually verifying all ballots. This example shows marks that were clearly intended as “No” votes, where the voter filled in the “o” in “No” rather than filling in the voting target (top). We found evidence that this is a replicable failure of ballot design: the “o” in “No” is mistaken for a voting target by a fraction of the voting public. The top two rows show all 10 examples of this phenomenon we found among the undervotes in this race. Some voters even filled in both the “o” in “No” as well as the voting target (bottom).

4.5.2 Applications

Our system lets a person efficiently interpret a large set of ballot images. As such, it can be used to detect and correct many common failure modes under typical-case assumptions,[5] such as mark-sense failures—failures of machines to correctly recognize marks made by the voter—and configuration failures—failures of machines to interpret the detected marks correctly (see Section 4.1).

The primary benefit of our approach is that it greatly increases the number of ballots an individual can reasonably inspect for him or herself, enabling each interested individual to arrive at his or her own interpretation of an entire contest. This can lead to interesting possibilities. For instance, because the output of our verification scheme is a classification—*from the user*—of every individual ballot, if multiple people examine the same contest, their classifications of individual ballots could be compared. Ballots for which the users' interpretations disagree could be easily discovered.

Another application of our approach may be to enable researchers to efficiently establish ground truth for evaluating novel opscan systems. For instance, researchers studying the accuracy of a new voter-intent classification scheme, or a stray-mark detector, could use our approach to test their system on an order-of-magnitude more human-verified ballot images than before.

Our approach may also make it feasible to do post-election logic and accuracy tests of current optical scan equipment by analyzing the equipment's interpretation of ballot images. Today, logic and accuracy testing typically uses a relatively small sample of test ballots, and test ballots are often marked in an unrealistically accurate fashion (e.g., test

[5]We make a distinction between *typical-case* assumptions in which the optical scan system may fail non-maliciously, for instance as a result of operator or programmer error or imperfect algorithms, and *worst-case* assumptions in which the scanning equipment and operators (among other stakeholders in the election) may act maliciously.

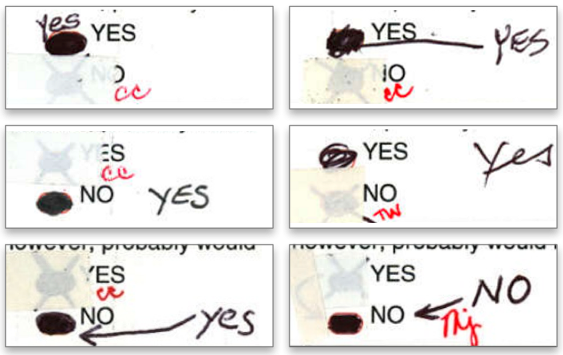


Figure 4.19. Examples of potentially ambiguous votes we found in our data set. In each example, the voter appears to be overriding or correcting his or her first mark. However, the interpretation of votes remains highly subjective. The initials in red (CC, TW, and MJ) are from voting officials who judged the voter intent and *enhanced* the ballots by covering some regions with tape. In each example, both voting targets are filled and one is crossed out; during the enhancement process, one of these targets was covered with tape and thus is only faintly visible in the images above. Because enhancement was performed before the ballots were scanned, we have only the scans of enhanced ballots, not the original ballots.

ballots may come pre-marked by the ballot printer, saving time but failing to anticipate and exhaust all the ways a voter will interact with a ballot). It would be possible for an election official to use our technique to verify the opscan system's interpretation of all ballot images, after the election, and use this to detect configuration errors and assess the system's accuracy. This may provide a better foundation for post-election logic and accuracy testing.

4.5.3 Limitations

Our approach is based on verifying ballot *images*, and is thus limited by the accuracy of this representation. In particular, our approach requires us to trust that the images are faithful and authentic scans of the paper ballots. As such, *we do not suggest our approach as a replacement for manual audits of paper ballots.* While one can envision auditing the scanning process to establish stronger trust in the images, we consider this an orthogonal issue and leave it as future work.

Working with ballot images poses other limitations as well. For instance, artifacts like ink bleed-through or damage to the paper ballot may be more difficult to recognize or diagnose from a scanned image than from the original paper ballot. In these cases, examining the physical paper ballot would remain important.

Some types of contests also pose issues for our approach. For example, a "vote-for-N" contest, in which the voter is allowed to mark more than one choice, complicates grouping

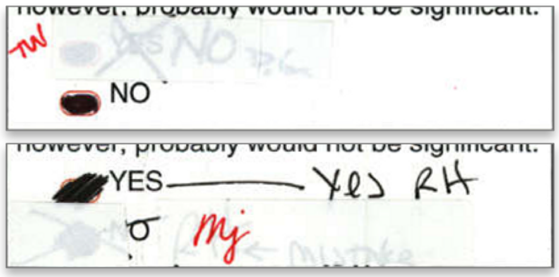


Figure 4.20. Examples of potentially identifying initials we discovered while verifying the election. The initials in black (DG, faintly visible on top after ballot enhancement, and RH on the bottom) appear to come from the voter. The initials in red (TW on top and MJ on the bottom) are from officials who judged voter intent.

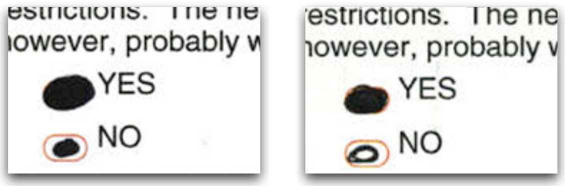


Figure 4.21. Two example anomalous ballots we discovered with our overlay approach that are *not* enhanced by voting officials, indicating that they went undetected. A potential explanation for this is that the opscan system detected these as "Yes" votes—rather than overvotes—triggering no further review of the ballots. (In our evaluation both votes were subjectively labeled "Yes" votes, after initially being classified as overvotes.)

ballots into a small number of visually similar groups that can be verified simultaneously.[6] While our system would still yield accurate results, its efficiency would be affected.

Similarly, issues such as ballot rotations—in which the order of choices may vary ballot by ballot—also complicate our approach. In this case, it is no longer safe to assume the position of a voting target corresponds to a particular choice or candidate, so ballots must be grouped taking into account the particular voting arrangement. However, because groupings are verified visually, this again only results in a loss of speed, and not correctness.

A specific limitation of our evaluation is that the ballots in our data set were *enhanced* by voting officials prior to scanning. In other words, many anomalous ballots discovered by the officials (e.g., overvotes rejected by the scanner) have been corrected either with felt pen or tape. While this prevented us from working with the original anomalies in those cases, we note that we can easily detect enhancements due to the initials that accompany them,[7] and our data set also contains anomalies that went undetected—or at least were not enhanced—by voting officials (see Figure 4.21).

[6]In the worst case there can be $n!$ combinations, where n is the number of candidates (though this is bounded by the total number of ballots).

[7]Enhancements are initialed by the official making the correction.

```
OVERLAY-IMAGE($min\_image, max\_image, min\_color, max\_color$)
1  $overlay\_image \leftarrow$ new $I \times J$ color image
2  for $j \leftarrow 0$ to $J - 1$
3  do for $i \leftarrow 0$ to $I - 1$
4      do $\alpha \leftarrow (max\_image_{i,j} - min\_image_{i,j}) \,/\, (255 - min\_image_{i,j})^{\dagger}$
5          $hue \leftarrow \alpha \cdot min\_color \; + \; (1 - \alpha) \cdot max\_color$
6          $\beta \leftarrow min\_image_{i,j} \,/\, 255$
7          $overlay\_image_{i,j} \leftarrow \beta \cdot$ white $+ \; (1 - \beta) \cdot hue$
8  return $overlay\_image$
```

Figure 4.22. Pseudocode for an alternative approach to create an overlay image from a pair of min- and max-images. Pixels in the resulting overlay image will vary in hue between *max-color* (i.e., blue in our examples) and *min-color* (i.e., red) depending on how far apart the darkest and lightest pixels are at a particular location. The whiteness of an overlay pixel depends on the intensity of the darkest pixel at a particular location. †In the case that $min\_image_{i,j} = max\_image_{i,j} = 255$, let $\alpha = 0$ (hue drops out anyway on line 7).

Another specific limitation of our evaluation is that the number of images given to us did not exactly match the number of ballots cast in the official canvass (43). The official canvass included 74 additional ballots not in our data set. This limited our ability to compare our human-verified results with the results of the official canvass.

4.5.4 Alternate overlay algorithm

Two requirements for an overlay image are that it capture the intensity of the min- and max-images, and that they be differentiable from one another. When we plot the spectrum of possible colors that result from our proposed algorithm we get the graph on the left in Figure 4.23. In particular, note the three corners of the spectrum that denote the common cases: the bottom-left corner is blue, corresponding to a region of foreground where all the pixels in the set are black; the top-right corner is white, corresponding to a background region where all pixels are white; and the top-left corner is red, corresponding to the case where at least one pixel is black and another is white, commonly because of a stray mark.

Another property of this spectrum to consider, however, is the interpolation of colors between corners. In particular, consider the edges. Ideally, along the top edge, we would like a smooth interpolation between red and white, which corresponds to regions that are white in at least one ballot, but dark in another—likely stray marks. The gradient on the diagonal, on the other hand, corresponds to regions that are identical in all ballots, even if they vary in shade between black and white. In this case, we would like a smooth interpolation between blue and white.

Our original overlay algorithm does not vary smoothly from blue to white along the diagonal, resulting in a more *conservative* overlay image: images that are possibly identical

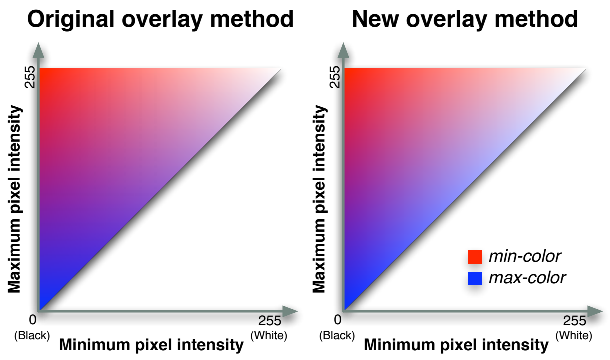


Figure 4.23. A comparison of the resulting range of colors a pixel may take, as produced by our two proposed OVERLAY-IMAGE algorithms. The image on the left is produced by the algorithm in Figure 4.8. The image on the right is produced by the algorithm in Figure 4.22. Note the differences in color, particularly along the diagonal: the left image shows a gradient from blue to purple to white, while the right image shows a gradient from blue to white.

may be perceived as containing more variation than present, requiring more work to verify.[8] To address this shortcoming, we have designed an alternative algorithm (see Figure 4.22) to ensure that the diagonal varies smoothly from blue to white, with no red, as we see in the graph on the right in Figure 4.23. Our experiments in Section 4.4 used the algorithm in Figure 4.8, but in hindsight the algorithm in Figure 4.22 may have been a better choice.

4.5.5 Future work and open problems

It is an interesting open problem to further reduce the time needed to verify a set of ballot images, without sacrificing accuracy. One possible improvement would be to implement a *clustering algorithm* when selecting ballots to overlay. Instead of placing ballots into groups randomly, we could cluster these images into groups of visually similar ballots. The hope is that if all ballots in a group are largely consistent in marking, it will be easier to verify the group of ballots, and it will be possible to increase the number of ballots in each group. In practice, we suspect the number of anomalous ballots will be small, and consistently marked ballots large, enabling the user to step through large groups quickly.

While this research focuses on verifying a set of ballot images, another area for future improvement is in the initial image analysis stage of our system. For example, it may be possible to automatically predict which ballots will need to be treated individually because they contain interesting anomalies. Similarly, machine learning techniques could be applied to better classify marginal or otherwise interesting marks. It is possible that analyzing the ballot images more thoroughly in advance may enable verification to proceed more efficiently.

[8]The difference between both algorithms is most evident on images with a lot of gray; in practice, though, both algorithms resulted in similar looking overlay images as our ballots contained mainly black and white features.

4.6 Related work

Others have proposed using scanned images of ballots to analyze and independently recount ballots. The Humboldt County Election Transparency Project rescanned all ballots cast in Humboldt County in several elections using a commercial off-the-shelf document scanner (42). Our evaluation relies upon a collection of ballot images scanned by the Humboldt County Election Transparency Project. We are indebted to them for making this data set available to us.

Mitch Trachtenberg developed BallotBrowser, open-source software to independently recount all contests by analyzing these ballot images. BallotBrowser introduces a novel technique to help the user quickly examine and verify BallotBrowser's interpretation of many marks, by displaying a cropped version of the voting target from many ballots simultaneously on the screen. This lets BallotBrowser display many marks on the screen at once, exploiting human parallelism. However, the user must still visually examine each such mark to gain confidence that all were properly interpreted. We superimpose ballot images to reduce the number of images that the user must examine. Like other optical scan systems, BallotBrowser relies on configuration information that tells it where on the ballot to look for voter marks and how to interpret each mark.

VotoScope is an early open-source software project for analyzing a collection of ballot images from an optical scan system to provide an independent tally of the ballots (45). Like other optical scan systems, both VotoScope and BallotBrowser are susceptible to mark-sensing and configuration failures. We were inspired and motivated by VotoScope and BallotBrowser. Our work is complementary to their work, as our techniques could be used to efficiently verify the results produced by their systems. In some cases, if the official optical scan system can export the set of all ballot images as well as its interpretation of those ballot images, our techniques could be applied directly to the output of the official opscan system, thus potentially taking the place of VotoScope or BallotBrowser.

Recently, the PERFECT project has pioneered the application of document analysis techniques to automated analysis and interpretation of optical scan ballots (84; 73; 55). Our work is distinguished from theirs largely by our focus on interactive tools (as opposed to entirely automated algorithms for interpreting ballot images) as well as our focus on verifying the accuracy of a set of ballot interpretations presented to us (rather than building algorithms to produce those interpretations in the first place).

Our work was also inspired by research on interactive computer vision (53; 76), which solves hard computer vision problems by letting the user of a tool give interactive hints to the algorithm. We take a similar approach: instead of attempting to perfectly infer the correct interpretation of every single ballot, we allow the user to interactively guide his or her examination of ballot images through the partitioning operations.

4.7 Conclusion

This chapter introduced techniques to help a human verify the interpretation of a large collection of ballot images, allowing the user to confirm that all ballots were interpreted accurately and to identify anomalies and ambiguous cases. While our approach requires trust in the ballot images and in the software displaying them, it has the benefit of significantly reducing the cost of recounts.

Our experiments suggest that the approach has promise. We found that our approach provides an order-of-magnitude improvement in speed, compared to a ballot-by-ballot recount. We also demonstrated that our approach is able to find a variety of anomalous ballots, many of which would have been undetected and possibly misinterpreted by an optical scan machine. These results suggest that our approach is practical and has the potential to improve the accuracy and transparency of elections.

Chapter 5

Conclusion

In addition to free and fair elections, democracy demands persuasive elections. Elections must be transparent to best persuade. They must be transparent enough to convince each stakeholder, with much to gain or lose, of the election's legitimacy. In this thesis, I presented three lines of research to improve the transparency of elections. In Chapter 2, I explored the problem of transparent and trustworthy random selections for election audits. In Chapter 3, I proposed a logging subsystem for electronic voting machines that can replay its inputs and outputs while balancing the requirement for voter secrecy. Finally, in Chapter 4, I developed a new approach for visually inspecting large numbers of ballot images efficiently and comprehensively.

The tension between anonymity and integrity, the enormous stakes, and the complex, non-transparent technology in use today make the voting problem hard. We can improve our confidence in our voting systems by auditing and investigating elections. However, for such post-election investigations to be meaningful, the procedures that support them, and the evidence that is recorded, must be transparent and trustworthy. Further, the investigations must be cost-effective to see regular use. The research I presented moves us in this direction.

Bibliography

[1] "California Elections Code," 1965, Section 19201.

[2] "California Elections Code," 1965, Section 336.5.

[3] "Bush v. Gore," vol. 531 U.S. 98, 2000, http://supreme.justia.com/us/531/98/case.html.

[4] December 2004, http://www.washingtonpost.com/ac2/wp-dyn/A64737-2004Dec14.

[5] "United states elections 2004," 2004, http://usinfo.state.gov/products/pubs/election04/procedure.htm.

[6] "Voluntary voting system guidelines," 2005, http://www.eac.gov/voting%20systems/voting-system-certification/2005-vvsg.

[7] "Summary of EAC Staff Level Meeting on Voting System Testing and Associated Costs," Testimony of Brian J. Hancock, Director, Testing and Certification; EAC Public Meeting May 17 2007, May 2007, http://www.eac.gov/News/docs/2007-meetings-and-hearings-5-17-07-meeting-publicmtg05-17-07estimonyhancock.pdf/attachment_download/file.

[8] B. Adida, "Helios: web-based open-audit voting," in *SS'08: Proceedings of the 17th conference on Security symposium.* Berkeley, CA, USA: USENIX Association, 2008, pp. 335–348. [Online]. Available: http://portal.acm.org/citation.cfm?id=1496734

[9] T. Antonyan, S. Davtyan, S. Kentros, A. Kiayias, L. Michel, N. Nicolaou, A. Russell, and A. Shvartsman, "Automating Voting Terminal Event Log Analysis," in *EVT'09: Proceedings of the conference on Electronic voting technology.* Berkeley, CA, USA: USENIX Association, 2009, http://www.usenix.org/events/evtwote09/tech/full_papers/antonyan.pdf.

[10] S. M. Bellovin, "Personal communication," August 2004.

[11] W. W. Berry and B. Lipari, "Public comment on the onondaga county cost comparison of paper ballot optical scan and direct recording electronic voting systems," 2007, http://www.nyvv.org/newdoc/county/CmtsOnondagaCostRpt.pdf.

[12] J. Bethencourt, D. Boneh, and B. Waters, "Cryptographic methods for storing ballots on a voting machine," in *In Proceedings of the 14th Network and Distributed System Security Symposium*, 2007, pp. 209–222.

[13] M. Blaze, A. Cordero, S. Engle, C. Karlof, N. Sastry, M. Sherr, T. Stegers, and K.-P. Yee, "Source Code Review of the Sequoia Voting System," July 2007, http://www.sos.ca.gov/elections/voting_systems/ttbr/sequoia-source-public-jul26.pdf.

[14] D. Bowen, "California Secretary of State Debra Bowen's Report to the Election Assistance Commission Concerning Errors and Deficiencies in Diebold/Premier GEMS Version 1.18.19," March 2009, http://www.sos.ca.gov/elections/voting_systems/sos-humboldt-report-to-eac-03-02-09.pdf.

[15] ——, "Secretary of State Debra Bowen Withdraws State Approval of Premier Voting System; Legislation to Require Disclosure of Product Flaws Clears First Hurdle," March 2009, http://www.sos.ca.gov/admin/press-releases/2009/DB09-017.pdf.

[16] ——, "Withdrawal of approval of premier election solutions, inc. diebold election systems, inc., gems 1.18.19," April 2009, http://www.sos.ca.gov/elections/ccrov/pdf/2009/april/09053rm.pdf.

[17] D. Bowen *et al.*, "Top-to-Bottom Review," August 2007, http://www.sos.ca.gov/elections/elections_vsr.htm.

[18] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman, "Scantegrity II: end-to-end verifiability for optical scan election systems using invisible ink confirmation codes," in *EVT'08: Proceedings of the conference on Electronic voting technology*. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–13.

[19] S. Checkoway, A. J. Feldman, B. Kantor, J. A. Halderman, E. W. Felten, and H. Shacham, "Can dre's provide long-lasting security? the case of return-oriented programming and the avc advantage," in *EVT'09: Proceedings of the conference on Electronic voting technology*. Berkeley, CA, USA: USENIX Association, 2009.

[20] J.-L. Chen and H.-J. Lee, "An efficient algorithm for form structure extraction using strip projection," *Pattern Recognition*, September 1998.

[21] E. V. Cross, G. Rogers, J. McClendon, W. Mitchell, K. Rouse, P. Gupta, P. Williams, I. Mkpong-Ruffin, Y. McMillian, E. Neely, J. Lane, H. Blunt, and J. E. Gilbert, "Prime III: One Machine, One Vote for Everyone," in *On-Line Proceedings of VoComp 2007*, July 2007, http://www.vocomp.org/papers/primeIII.pdf.

[22] David L. Dill and Dan S. Wallach, "Florida's district 13 election in 2006: Can statistics tell us who won?" 2008, http://www.mathaware.org/mam/08/AshFlorida.pdf.

[23] Democratic National Committee: Institute of Voting Rights, "Democracy At Risk: The 2004 Election in Ohio," 2005, http://www.democrats.org/pdfs/ohvrireport/section02.pdf.

[24] D. L. Dill and D. S. Wallach, “Stones unturned: Gaps in the investigation of sarasota’s disputed congressional election,” April 2007, http://www.cs.rice.edu/~dwallach/pub/sarasota07.pdf.

[25] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, “Xen and the art of virtualization,” in *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003. [Online]. Available: citeseer.ist.psu.edu/dragovic03xen.html

[26] D. Eastlake, “Publicly Verifiable Nominations Committee (NomCom) Random Selection,” June 2004, rFC 3797.

[27] Encyclopaedia Britannica, “dice,” April 2006, Encyclopaedia Britannica Premium Service.

[28] S. P. Everett, “The usability of electronic voting machines and how votes can be changed without detection,” Ph.D. dissertation, Rice University, 2007.

[29] A. J. Feldman, J. A. Halderman, and E. W. Felten, “Security analysis of the diebold AccuVote-TS voting machine,” in *EVT’07: Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*. Berkeley, CA, USA: USENIX Association, 2007.

[30] P. Fimrite, “Surfer-mayor wannabe apparently wipes out; Judge disqualifies write-in votes with bubbles unshaded,” *San Francisco Chronicle*, November 2004, http://www.sfgate.com/cgi-bin/article.cgi?f=/c/a/2004/11/23/BAGM9A02VA1.DTL.

[31] B. Friedman, “Two ohio election officials convicted for rigging 2004 presidential recount!” *Brad Blog*, January 2007, http://www.bradblog.com/?p=4071.

[32] E. Gal and S. Toledo, “Algorithms and data structures for flash memories,” *ACM Computing Surveys*, vol. 37, no. 2, 2005.

[33] A. Garber and D. Postman, “Democrats will finance a statewide hand recount,” *The Seattle Times*, December 2004, http://seattletimes.nwsource.com/html/localnews/2002108596_webgov03.html.

[34] S. Garera and A. D. Rubin, “An independent audit framework for software dependent voting systems,” in *14th ACM conference on Computer and Communications Security*, 2007.

[35] C. Gilbert, “Coleman Concedes to Franken in MN Senate Race,” *Minnesota Public Radio*, December 2008.

[36] J. A. Halderman, “Diebold shows how to make your own voting machine key,” January 2007.

[37] J. L. Hall, “Policy mechanisms for increasing transparency in electronic voting,” PhD Dissertation, University of California at Berkeley, Information Management and Systems, 2008.

[38] J. L. Hall, P. B. Stark, L. W. Miratrix, M. Briones, E. Ginnold, F. Oakley, M. Peaden, G. Pellerin, T. Stanionis, and T. Webber, "Implementing Risk-Limiting Post-Election Audits in California," in *Electronic Voting Technology (EVT'09).* Berkeley, CA, USA: USENIX Association, 2009, http://www.usenix.org/event/evtwote09/tech/full_papers/hall.pdf.

[39] B. Harris, *Black Box Voting: Vote Tampering in the 21st Century.* Elon House/Plan Nine, July 2003.

[40] J. Hartline, E. Hong, A. Mohr, W. Pentney, and E. Rocke, "Characterizing history independent data structures," in *International Society for Analysis, its Applications and Computation*, 2002.

[41] J. Holguin, "E-voting: Is the fix in?" 2004, http://www.cbsnews.com/stories/2004/07/28/sunday/main632436.shtml.

[42] "Humboldt County Election Transparency Project," http://humtp.com.

[43] Humboldt County Elections and Voter Registration, "Elections and Voter Registration," June 2008, http://co.humboldt.ca.us/election/content/content.asp?pg=election_results.htm.

[44] H. Hursti, "The Black Box Report Critical Security Issues with Diebold Optical Scan Design," July 2005, http://www.blackboxvoting.org/BBVreport.pdf.

[45] ——, "Votoscope software," October 2005, http://vote.nist.gov/comment_harri_hursti.pdf.

[46] S. Inguva, E. Rescorla, H. Shacham, and D. S. Wallach, "Source Code Review of the Hart InterCivic Voting System," July 2007, http://www.sos.ca.gov/elections/voting_systems/ttbr/Hart-source-public.pdf.

[47] D. Jefferson, E. Ginnold, K. Midstokke, K. Alexander, P. Stark, and A. Lehmkuhl, "Evaluation of Audit Sampling Models and Options for Strengthening California's Manual Count," July 2007, http://www.sos.ca.gov/elections/peas/final_peaswg_report.pdf.

[48] K. C. Johnson, "Election certification by statistical audit of voter-verified paper ballots," October 2004, available at SSRN: http://ssrn.com/abstract=640943.

[49] D. W. Jones, "A Brief Illustrated History of Voting," 2003, http://www.cs.uiowa.edu/~jones/voting/pictures.

[50] ——, *Towards Trustworthy Elections*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6000/2010, chapter On Optical Mark-Sense Scanning, pp. 175–190.

[51] J. Kilian, *Uses of Randomness in Algorithms and Protocols.* MIT Press, 1990.

[52] T. Kohno, A. Stubblefield, A. Rubin, and D. Wallach, “Analysis of an electronic voting system,” in *IEEE Symposium on Security and Privacy*, 2004.

[53] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum, “Lazy snapping,” in *SIGGRAPH ’04: ACM SIGGRAPH 2004 Papers*. New York, NY, USA: ACM, 2004, pp. 303–308.

[54] S. Lloyd, “Least-Square Quantization in PCM,” in *IEEE Transactions on Information Theory*, vol. 28, no. 2, 1982, pp. 129–137.

[55] D. Lopresti, G. Nagy, and E. B. Smith, “Document Analysis Issues in Reading Optical Scan Ballots,” in *DAS ’10: Proceedings of the 8th IAPR International Workshop on Document Analysis Systems*. New York, NY, USA: ACM, 2010, pp. 105–112.

[56] J. Mazzolini, “Workers accused of fudging ’04 recount,” The Cleveland Plain Dealer, 4 2006.

[57] P. McDaniel, K. Butler, W. Enck, H. Husti, S. McLaughlin, P. Traynor, M. Blaze, A. Aviv, P. Cerny, S. Clark, E. Cronin, G. Shah, M. Sherr, G. Vigna, R. Kemmerer, D. Balzarotti, G. Banks, M. Cova, V. Felmetsger, W. Robertson, F. Valeur, J. L. Hall, and L. Quilter, “Everest: Evaluation and validation of election-related equipment, standards and testing,” December 2007, http://www.sos.state.oh.us/SOS/upload/everest/14-AcademicFinalEVERESTReport.pdf.

[58] D. Molnar, T. Kohno, N. Sastry, and D. Wagner, “Tamper-Evident, History-Independent, Subliminal-Free Data Structures on PROM Storage -or- How to Store Ballots on a Voting Machine (Extended Abstract),” in *IEEE Symposium on Security and Privacy*, 2006.

[59] M. Naor and V. Teague, “Anti-persistence: History independent data structures,” in *Symposium Theory of Computing*, 2001.

[60] C. A. Neff, “Election confidence,” Manuscript, 2003.

[61] Office of the Onondaga County Comptroller, “Cost comparison of optical scanning and digital recording electronic devices,” 2007.

[62] Perry Bacon Jr., “Coleman Concedes to Franken in MN Senate Race,” *The Washington Post*, June 2009, http://voices.washingtonpost.com/capitol-briefing/2009/06/minn_supreme_court_declares_fr.html.

[63] “Pygame,” http://www.pygame.org/.

[64] Rand Corporation, Ed., *A Million Random Digits with 100,000 Normal Deviates*. Rand Corporation, 1955.

[65] E. Rescorla, “Understanding the security properties of ballot-based verification techniques,” in *Electronic Voting Technology (EVT’09)*. USENIX Association, 2009, http://www.usenix.org/event/evtwote09/tech/full_papers/rescorla-ballot.pdf.

[66] R. Rivest, "The threeballot voting system," October 2006, http://people.csail.mit.edu/rivest/Rivest-TheThreeBallotVotingSystem.pdf.

[67] T. Rohwer, "Faulty Voting Machines Delay Results; Counting Under Way," *The Daily Nonpareil Online*, June 2006.

[68] P. Y. A. Ryan, "The computer ate my vote," University of Newcastle upon Tyne, Tech. Rep. CS-TR-988, 2006.

[69] D. Sandler and D. S. Wallach, "Casting votes in the auditorium," in *USENIX/Accurate Electronic Voting Technology Workshop*, 2007.

[70] N. Sastry, T. Kohno, and D. Wagner, "Designing voting machines for verification," in *USENIX Security Symposium*, 2006.

[71] B. Schneier, *Applied Cryptography*, 2nd ed., P. Sutherland, Ed. John Wiley & Sons, Inc., 1996.

[72] A. Singh, "Is virginia ready for election day," October 2008, http://www.wset.com/news/stories/1008/563646.html.

[73] E. H. B. Smith, D. Lopresti, and G. Nagy, "Ballot Mark Detection," *19th International Conference on Pattern Recognition*, 2008.

[74] P. Smith and B. Kibrick, "Manual Audit Requirements," Verified Voting Foundation, September 2005.

[75] N. Starr, "Nonrandom risk: The 1970 draft lottery," *Journal of Statistics Education*, vol. 5, no. 2, 1997.

[76] J. Sun, J. Jia, C.-K. Tang, and H.-Y. Shum, "Poisson matting," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 315–321, 2004.

[77] E. Theisen, "Vendors are undermining the structure of u.s. elections," August 2008, http://www.votersunite.org/info/ReclaimElections.pdf.

[78] M. Trachtenberg, "Ballot browser," 2008, http://www.mitchtrachtenberg.com/BallotBrowser.

[79] Verified Voting Foundation, "The Verifier," 2008, http://www.verifiedvoting.org/verifier.

[80] D. Wagner, J. A. Calandrino, A. J. Feldman, J. A. Halderman, H. Yu, and W. P. Zeller, "Source Code Review of the Diebold Voting System," July 2007, http://www.sos.ca.gov/elections/voting_systems/ttbr/diebold-source-public-jul29.pdf.

[81] D. A. Wheeler, "SLOCCount User's Guide," 2004, http://www.dwheeler.com/sloccount/sloccount.html.

[82] P. Williams, E. V. Cross, I. Mkpong-Ruffin, Y. McMillian, K. Nobles, P. Gupta, and J. E. Gilbert, “Prime III: Where Usable Security and Electronic Voting Meet,” February 2007, http://www.usablesecurity.org/papers/primeIII.pdf.

[83] R. Wolf, “Get out the pencils: Paper ballots return,” *USA Today*, February 2008, http://www.usatoday.com/news/politics/election2008/2008-02-28-votinginside_N.htm.

[84] P. Xiu, D. Lopresti, H. Baird, G. Nagy, and E. B. Smith, “Style-Based Ballot Mark Recognition,” 2009.

[85] A. Yasinsac, D. Wagner, M. Bishop, T. Baker, B. de Medeiros, G. Tyson, M. Shamos, and M. Burmester, “Software Review and Security Analysis of the ES&S iVotronic 8.0.1.2 Voting Machine Firmware,” February 2007, http://www.cs.berkeley.edu/~daw/papers/sarasota07.pdf.

[86] K. Yee, “Building reliable voting machine software,” Ph.D. dissertation, UC Berkeley, 2007.

[87] K.-P. Yee, “Report on the Pvote security review,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2007-136, Nov 2007. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-136.html

[88] K. Zetter, “Diebold Admits Systemic Audit Log Failure; State Vows Inquiry,” March 2009, .

[89] ——, “Voting System Adds Nearly 5,000 Ballots to Tally,” June 2009, http://www.wired.com/threatlevel/2009/06/voting-machine-adds-nearly-5000-ballots-to-tally.