# Software Support for Software-Independent Auditing — Short Paper

Gabrielle A. Gianelli, Jennifer D. King, Edward W. Felten, and William P. Zeller

Center for Information Technology Policy
Department of Computer Science
Princeton University
{gianelli,jdking,felten,wzeller}@princeton.edu

## Abstract

Electronic voting machines have the potential to make the election process more efficient, but concerns over their reliability and security could undermine confidence in election results. The most effective way of verifying election results is by auditing physical copies of the ballots that have been verified by the voter. Since manually viewing every ballot is unrealistic, a variety of algorithms can be used to determine which ballots or precincts to audit. These algorithms generate a list of ballots (or precincts) to audit and declare the election outcome correct (with a certain confidence level) if the audited physical ballot matches the corresponding electronic records. Many of these algorithms are complicated and require a computer on which to run. Since this computer is susceptible to the same issues as electronic voting machines, we want to be able to trust the output of these algorithms without trusting the computer on which the auditing algorithms run. We created an online auditing system with three goals in mind. First, we ensure software-independence by publishing a log that allows anyone to verify (either manually or by using a computer) that the audit procedures were followed correctly. This allows the computer running the auditing algorithms to be untrusted. Second, we implemented recently published auditing algorithms that reduce the number of precincts or ballots to be inspected. Third, we provide a user-friendly interface that can be used by ordinary government officials. Additionally, the modular design of our system does not restrict the algorithms that can be used, allowing it to satisfy the variety of legal requirements in different states.

## 1 Introduction

When electronic voting machines are used in elections, the correctness of the outcome depends on the correctness of these machines. In order to prevent machine failure or corruption from influencing the outcome of an election, voter-verified paper auditing trails (VVPATs) can be used to confirm that the electronic results match the verified paper ballots. (We use the term "VVPAT" to describe both optical-scan ballots and Direct-Recording Electronic (DRE) paper trails.) This verification is done by an audit that requires a manual inspection of some number of the ballots used in the election.

Many algorithms exist for choosing which ballots to inspect manually. These algorithms vary in their complexity, their method of selecting ballots for manual auditing, and the number of ballots they require to be inspected to guarantee a given level of confidence in the election.

An electronic auditing system could be used to carry out these algorithms, but this approach would create a need for yet another layer of verification.

In order to avoid having to trust any auditing machine, we implemented an electronic auditing system that logs all of the information necessary to allow an independent system to recreate and verify the selection of ballots to be audited. This information includes pseudorandom numbers generated using a variant of "Dice Plus CSPRNGS" of Calandrino *et al.* [5].

Our system achieves software independence [12] because its log allows any third-party to check the correctness of our system either by hand or by using its own verification software. Our design follows the principle in [12]: "One should strongly prefer any approach where the integrity of the election outcome is not dependent on trusting the correctness of complex software". To show that the logs could be verified by a third-party, we implemented a prototype log verifier as a separate program.

We implemented this system as a website with an intuitive interface that allows a non-technical user to run a variety of auditing algorithms. As a part of this website, we implemented four common algorithms and designed the system to allow additional algorithms to be easily added. Our system, as implemented, is compatible with the voting regulations of many US states and

could easily be extended.

Our paper is organized as follows. Section 2 discusses the problems with current election technology and auditing procedures. In Section 3, we walk through the steps of the auditing process. In Section 4, we discuss the importance of keeping a verifiable log of all electronic calculations to ensure that the machine running the audit has not been compromised. In Section 5, we discuss the implementation of our system. In Section 6 we discuss our system's performance auditing unofficial November 2008 election data from a California county.

## 2 Background

### 2.1 Flaws in Voting Devices

The use of electronic voting machines raises security and privacy issues. These systems are general purpose computers, which means they are capable of running any program, whether benign or malicious. Due to the complexity of these systems, the software itself might contain bugs that could change the outcome of an election. An attacker could also compromise these machines. Certain electronic voting systems store election results in their own internal memory, which can be silently erased or manipulated. Additionally, many of these electronic systems share data on election day, possibly allowing an infected computer to spread a virus to other machines. Researchers have discovered vulnerabilities in a variety of voting systems, ranging from major security design flaws to minor, but exploitable, bugs.

Numerous research studies have found security flaws in electronic voting systems. For example, Feldman *et al.* [6] describe the ease with which they were able to compromise the Diebold AccuVote-TS, a widely used electronic voting system at that time [6]. The authors were able to quickly obtain access to the memory slot on the side of the machine and insert their own memory card containing malicious code which secretly flipped votes. They also described how a virus could be designed to spread automatically between voting machines to manipulate votes or to disable the machines entirely, which could change election results on a large scale or entirely shut down precincts.

These vulnerabilities are not limited to Diebold machines. A 2007 study commissioned by California's Secretary of State found similar security flaws in voting machines made by two other major companies [3].

### 2.2 Safeguards Against Election Fraud

Many researchers promote the use of physical paper ballots as a way to prevent and detect some types of voting fraud [10, 4, 6, 9, 7]. (While the use of optical scan machines is typically recommended, many DRE machines are able to generate a paper trail. Again, we use the term VVPAT to describe both optical scan ballots and DRE paper trails.) In a system that uses VVPAT, voters must be able to check that a physical copy of their vote matches how they voted. These records are kept for post-election auditing. Some electronic voting systems, such as optical scan machines, automatically generate paper records in their normal course of operation (the voter records her vote directly on a paper ballot before it is scanned). However, many DRE machines print either no paper record, or only print a report at the end of the day summarizing the vote totals on that machine. Ideally, if a DRE machine is to be used, it should print a paper record after each ballot is cast and allow the voter to inspect that paper before exiting the voting booth [10, 7].

The availability of physical copies of the ballots provides a layer of redundancy that makes fraud more difficult because of the differences between paper and electronic media. Feldman *et al.* [6] write, "the real advantage of a paper trail is that its failure modes differ significantly from those of electronic systems, making the combination of paper and electronic record-keeping harder to defraud than either would be alone…". Even if additional electronic copies were kept, there would be no guarantee that whatever problem affected the first copy of electronic records did not have an impact the second. Furthermore, an audit of the paper records can be rechecked by humans, allowing officials to be sure that the auditing software was not compromised as well [6].

### 2.3 Current Audit Legislation

Each state currently has its own election auditing laws. In 2007, fifteen states required manual recounts [8]. Often the laws dictate that a certain percentage of all precincts must undergo manual inspection, but this percentage can vary widely. For instance, California requires an audit of only 1% of precincts, whereas Connecticut requires 10% [8, 2].

When the precincts chosen by this method are large, the sizable number of ballots that must be audited can present a major challenge to the auditing process. After observing the Connecticut audits after the November 2007 election, the Connecticut Citizen Election Audit Coalition observed that "many of the audits, as observed, [left them] uncertain as to whether an error or fraud would have been detected in an audited race in this election," largely due to inconsistencies in the auditors' individual counting techniques and the lack of transparency that prohibited any public audit of the recount[13].

Since human error is an integral factor in the correctness of manual audits, reducing the number of ballots

that must be counted could reduce its effect. Statistical algorithms, such as those discussed in Section 5.2, can reduce the number of ballots selected while guaranteeing a high confidence level that fraud will be uncovered if it does exist. By reducing the total number of ballots that must be recounted, the accuracy of hand recounts would increase and less manpower would be needed to execute them. Thus, statistical algorithms are a useful tool for post-election auditing, though the implementation may be too complicated to do by hand.

# 3 Overview of the Auditing Procedure

The system we implemented was designed to handle the lack of uniformity between state auditing laws. Any number of algorithms can be added to the system, allowing a single, consistent interface to be used in states with different requirements. The following steps, also shown in Figure 1, would be performed during any audit on our system.

## 3.1 Administrative Set Up

The ballot information, such as the lists of precincts, races, and candidates, and the votes recorded on each ballot must be uploaded into the database. The system includes support for multiple elections, multiple races, and multiple winners within a given race. It is also possible to audit races that do not have individual ballots available using the precinct-based algorithms, as long as vote totals from each precinct can be uploaded.

If our system were to be adopted, we imagine tools would be created to automatically convert ballot data from currently existing vote tabulation systems. However, because many of these formats are undocumented, we did not implement these tools ourselves. For the test audits we ran, we converted the ballot data into a format suitable for our database by creating a few simple scripts.

## 3.2 User Log-In and Dashboard

After the system is initialized, a user (e.g., an election official) can login and view her main page, where she is presented with a dashboard that allows her to view a summary of all audits that she has performed and each audit's current status. She also has the option to resume any incomplete audits at the step where she was interrupted.

When starting a new audit, the user can either read a description of the audit's steps or immediately begin the audit process. The first page lists the name and date of each election the user is authorized to audit. The user selects the election to audit and decides whether to "link" the precincts[1].

If precincts are linked, then the algorithms will give preference to precincts that have already been selected for auditing for a previous race that is part of the same election. Linking precincts does not affect the auditing probabilities. This feature reduces the overall number of precincts where manual inspections are performed for a given election.

## 3.3 Race and Algorithm Selection

Once the user has selected an election, she is prompted to choose a race in the election and an algorithm to use on that race. A drop-down description of each algorithm appears when it is selected, along with a box for the user to enter her desired confidence level or probability, depending on the selected algorithm. The process of selecting a race and an associated algorithm is repeated until algorithms have been chosen for all races.

## 3.4 Pseudorandom Number Generation

In order to introduce pseudorandomness into the system, we follow the method described by Calandrino *et al.* [5], which asks the user to physically roll dice and enter a sequence of die rolls. These die rolls later are used as a seed for all pseudorandom number generation. Our system provides the option of using either six- or twenty-sided dice, which would require fifty or thirty rolls, respectively, in order to generate a 128-bit random seed.

Pseudorandom number generation must be done after all data has been uploaded into the system, and after all choices have been made by election officials. This prevents a potentially corrupt official from altering aspects of the data (such as re-ordering or re-naming precincts) in order to affect the order or manner in which precincts are processed. It is only after election officials have

[1] "Linking" precincts is an optimization, used in elections with multiple races, in which ballots that are audited for one race are preferentially chosen for auditing in another race, in a way that preserves the required statistical confidence guarantees. For example, suppose that there are two races, and the auditing algorithms require auditing of 15 randomly chosen precincts for Race A, and 10 randomly chosen precincts for Race B. Without linking, the precincts to be audited in Race A would be chosen independently of those audited in Race B. With linking, we would randomly choose 10 precincts to be audited in both races, and randomly choose five additional precincts to be audited in only Race A. This would meet the required statistical confidence in both races, at lower cost than in the unlinked case. A possible drawback of linking is that the possibility of an auditing "failure" (i.e., of the auditing algorithm failing to catch a significant error due to rare bad luck in choosing where to audit) in Race A is correlated with the possibility of failure in Race B. Whether this correlation matters is a value judgment beyond the scope of this paper. For more information on our implementation of linking, see Appendix B.

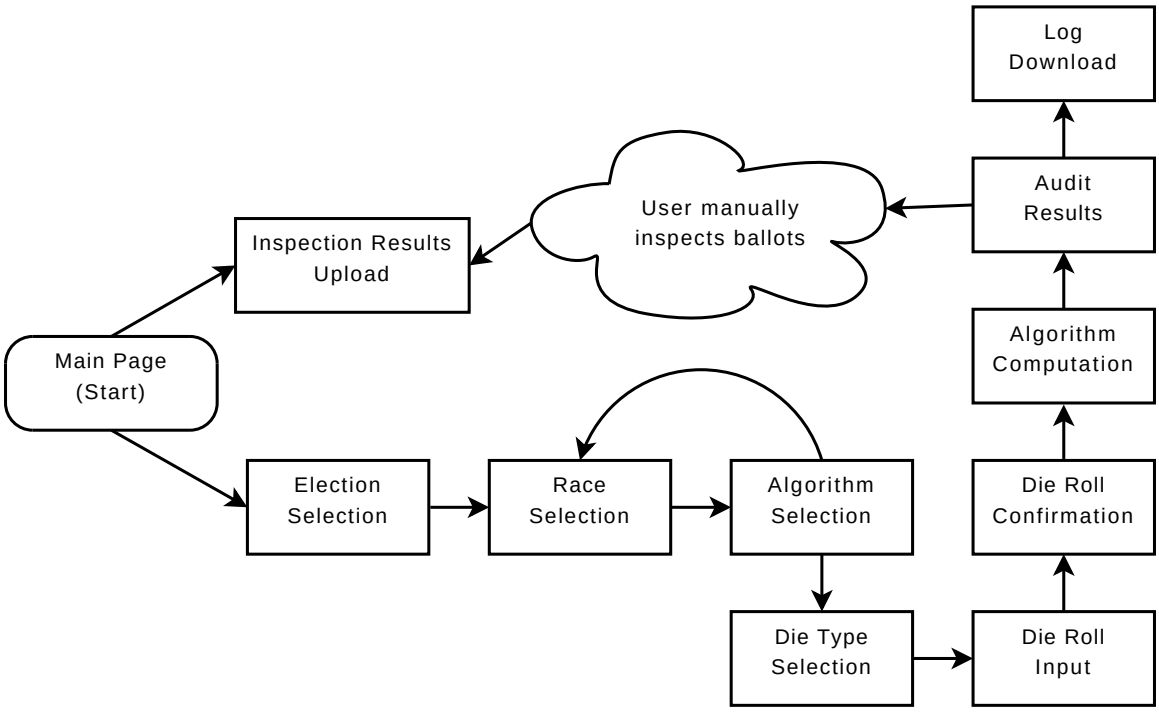


Figure 1: Flowchart of web page interaction.

made all of their discretionary choices that pseudorandom number generation starts.

It is essential for audit security that the random information be created by a truly stochastic process, which makes it necessary for the user, not the computer, to provide the initial string of bits, and for the die rolling to be done in public after the election, in the presence of observers.

### 3.5 Running the Algorithms

Once the user has entered the die roll information, she is presented with a page that displays the progress of each algorithm. After the algorithms have completed running in a background thread, the site automatically forwards the user to the results page.

### 3.6 Audit Results

The user can download or inspect her audit results in a log file, formatted in XML. This file contains a list of precincts and ballots to be audited, as well as all the logging information generated by the system in the course of the auditing process.

### 3.7 Ballot Inspection

At this point, the user should log out of the system and perform the specified manual inspections in accordance with the appropriate laws. When this process is completed, she should log back into the system to enter the results of the manual inspections. After entering this information, the page will display whether the inspection totals match those stored in the database. If the selected algorithm provides deterministic escalation criteria, the system can evaluate the outcome of the audit and proceed with escalation if a discrepancy exists. Otherwise, the user can decide whether the discrepancy in vote totals is enough to merit concern and can take the appropriate actions, according to the applicable election laws and procedures.

## 4 Log

### 4.1 Generating the Log

The log that the program generates is the most critical part of our auditing system. Our log records every detail needed to recreate the entire audit with a third-party system, allowing the audits to be verified by other computers or by hand. This feature provides the software independence that is necessary for users to have confidence in any machine-assisted auditing system.

The log is organized so that general information is stored first (e.g., the date and time of the audit, and the election name and date). The sequence of die rolls entered by the user and whether the user has chosen to link precincts are also stored in this section.

The next section contains all of the inputs for each algorithm and the associated outputs. These records allow a verification program to run each of the algorithms again using the logged input and verify that its output matches the output in the log. All outputs can be predicted because the seed of the pseudorandom generator is stored as the input.

For convenience, the final section of the log compiles the results of all the algorithms that have been run and lists the precincts and/or ballots that must be audited. This allows election officials to have a single list of the results of the algorithms without duplicates. The user can choose either to download this XML file or view it with an XSL template, which allows the user to more easily read the contents.

### 4.2 Verifying the Log

As we have said, the log file should be both human and computer readable so that the software's audit process can be verified. The use of XML allows a computer to parse this data with relative ease. We also implemented a prototype log verifier which parses the log file, runs each of the algorithms again based on the input recorded in the log file, and checks its own results against the output sections of the log. This module was designed to serve as a template for other log parsers.

In practice, any organization or member of the public would be free to verify the logs themselves.

## 5 Implementation

### 5.1 Programming Tools

We implemented our project modules in Python using a MySQL database. The schema is represented in Figure 2. The web interface uses the Django web development framework [1].

The system was written to be as modular as possible, with special emphasis on the separation between the database implementation and the algorithm module. This separation allows new algorithms to be added, as long as they implement the interface we define. The current version of our system contains four algorithms, all of which are unit tested in order to guarantee a certain level of confidence in our implementation.

### 5.2 Auditing Algorithms

In our system, we implemented four algorithms. The first two, Exact Percent (EP) and Percent by Probability (PBP) choose precincts for auditing based on a user defined percentage. The EP algorithm selects precincts at random, guaranteeing that at least the defined percentage of all precincts is chosen. The PBP algorithm is derived from a paper by Rivest, in which each precinct is chosen independently with the probability determined by the audit official [11].

We also implemented two other algorithms, proposed by Calandrino *et al.* [4], which select individual ballots for manual auditing. These are referred to as Constant Sample Size (CSS) and Varying Sample Size (VSS).

CSS chooses enough ballots across all precincts in a given race to guarantee with a user-defined confidence level that a fraudulent ballot will be chosen, if one exists. VSS is similar to CSS but also takes into account the fact that different precincts have different numbers of ballots. Because VSS chooses precincts first and ballots only from those precincts, it decreases the number of precincts that election officials may have to visit.

The CSS and VSS algorithms can provide huge benefits in comparison to other algorithms[2]. In practice, however, these methods are complicated to implement, and the computations can be difficult for nontechnical election officials to carry out manually. A software system like ours makes such algorithms practical.

## 6 Evaluation Using Humboldt County Data

To test our system on a realistic data set, we obtained unofficial data from the November 2008 elections in Humboldt County, CA, publicly available at [15]. The data consists of text files representing all the votes recorded on each ballot and scans of the physical ballots. The text files were generated by Ballot Browser, a publicly available tool that converts the ballot images to a textual representation [14]. Because of this format, the data was easily parsed into a form that was compatible with our system.

We ran three sample audits on the data to compare the results of the algorithms and simulate an actual audit. All of the audits spanned 145 precincts containing 128,144 ballots.

In the first audit, for each of the 29 races, we selected 1% of the precincts using the Exact Percent algorithm, in accordance with the requirements of California law. Thirty-three precincts were chosen for auditing, which

[2]Calandrino *et al.* [4] calculated that using their methods reduced the number of ballots that needed to be manually audited from 22% to 0.06% while maintaining a 99% confidence level for one election.

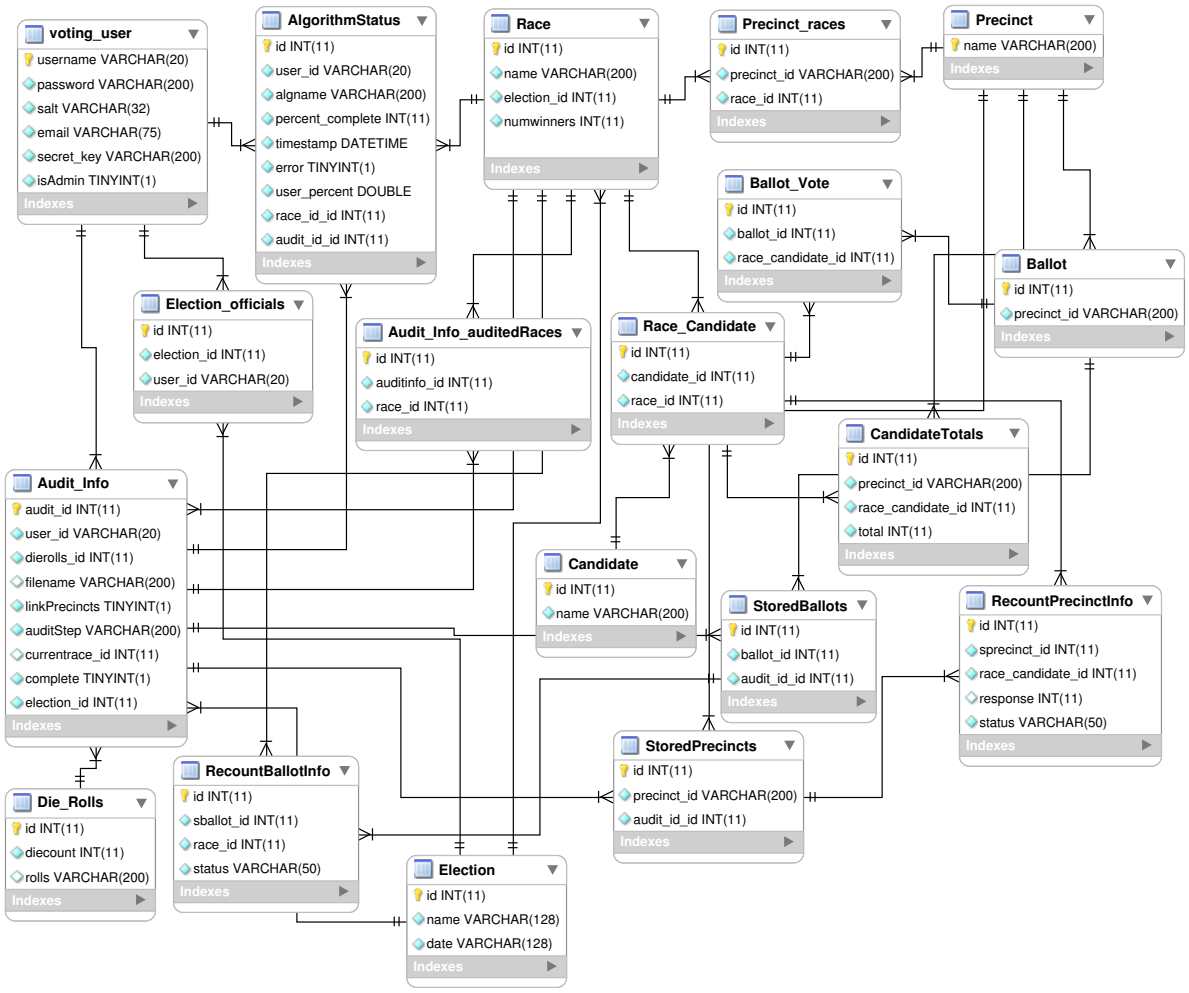


Figure 2: The database schema for our prototype. Tables created by Django have been removed.

amounts to 15,613 ballots, 12% of the total number of ballots. In the second trial, we used the same algorithm with the same percentage, but we linked precincts as described in Section 1. In this audit, the number of precincts selected was 15, less than half the number selected in the first trial, and altogether these precincts contained 5,768 ballots, or 4% of the total.

In the third audit, we used the Constant Sample Size algorithm, described in Section 5.2. This algorithm chose 3,006 ballots (2% of the total) for all 29 races, approximately half as many selected by EP with linked precincts and about one-eighth of the number chosen by EP without linked precincts. Manually checking the image files for every selected ballot was infeasible, but we manually confirmed the results for the Ferndale Mayor race, in which twelve ballots were selected for auditing using the CSS algorithm. Every vote on each of the twelve ballots we audited matched the corresponding record in the database.

## 7 Conclusion

The system we implemented provides an easy to use, extensible, software-independent vote auditing framework. While we implemented four different auditing algorithms, the modularity of the system allows new algorithms to be easily added, making our system amenable to any state's auditing requirements. The extensive logging we perform allows every meaningful action our system performs to be verified and replicated by any third party.

By providing a simple and trustworthy way of performing audits, we hope to encourage the use of paper records as a means of verifying the correctness of electronic voting machines. Electronic voting technology

will continue to be used across the country, but physical copies of ballots are necessary to ensure that these machines provide a fair and accurate record of the way people vote.

## References


[1] Django. http://www.djangoproject.com.

[2] Title 9–Elections, Sec. 9-320f. In *General Statutes of Connecticut*, chapter 158. 2009.

[3] M. Bishop. Overview of Red Team Reports. http://www.sos.ca.gov/elections/voting_systems/ttbr/red_overview.pdf.

[4] J.A. Calandrino, J.A. Halderman, and E.W. Felten. Machine Assisted Election Auditing. *USENIX/ACCURATE Electronic Voting Technology Workshop '07*, 2007.

[5] Joseph A. Calandrino, J. Alex Halderman, and Edward W. Felten. In defense of pseudorandom sample selection. *USENIX/ACCURATE Electronic Voting Technology Workshop 2008*, July 2008.

[6] A.J. Feldman, J.A. Halderman, and E.W. Felten. Security Analysis of the Diebold AccuVote-TS Voting Machine. *USENIX/ACCURATE Electronic Voting Technology Workshop '07*, 2006.

[7] E.A. Fischer. Election Reform and Electronic Voting Systems (DREs): Analysis of Security Issues. *CRS Report for Congress*, 2003.

[8] Post-Election Audit Standards Working Group. Evaluation of Audit Sampling Models and Opinions for Strengthening California's Manual Count. http://www.sos.ca.gov/elections/peas/final_peaswg_report.pdf.

[9] T. Kohno, A. Stubblefield, A.D. Rubin, and D.S. Wallach. Analysis of an Electronic Voting System. *IEEE Symposium on Security and Privacy 2004*, May 2004.

[10] Rebecca Mercuri. Explanation of Voter-Verified Ballot Systems. *ACM Software Engineering Notes (SIGSOFT)*, 27(5), 2002.

[11] R.L. Rivest. On Auditing Elections When Precincts Have Different Sizes. *Electronic Voting Technologies 2008)*, 2008.

[12] R.L. Rivest and J.P. Wack. On the notion of "software independence" in voting systems. http://vote.nist.gov/SI-in-voting.pdf, July 2006.

[13] The Connecticut Citizen Election Audit Coalition. Report and Feedback: November 2007 Connecticut Election Audit Observation. http://www.ctelectionaudit.org/Reports/AuditObservationReport.pdf.

[14] M. Trachtenberg. Ballot Browser. http://www.TEVSystems.com.

[15] M. Trachtenberg. Humboldt County November 2008 Election: Completely Unofficial Results. http://www.mitchtrachtenberg.com/Nov2008/index.html.


## A Algorithms

### A.1 Constant Sample Size (CSS)

CSS chooses a constant number of ballots to be audited across all the precincts in a given race, given a statistical confidence level. For the outcome of any race to change, the minimum number of ballots that must be changed is equal to half the difference in votes between the just winning candidate and the just losing candidate (e.g., in a race with one winner, the candidates with the top two vote totals; in a race with two winners, the candidates with the second and third top vote totals, etc.). This number of ballots, $B$, is the number of bad ballots that must exist in the election for any fraud to affect the outcome. Using the hypergeometric distribution, the algorithm finds a sample size $n$ ballots which must be selected across all the precincts in the race, and those $n$ ballots are selected from the full pool by shuffling the list of ballots and selecting the first $n$ ballots to be manually audited.

### A.2 Varying Sample Size

The second Calandrino algorithm selects precincts nonuniformly, with larger precincts more likely to be chosen, according to a formula derived by Calandrino. It then draws ballots to be manually audited from each chosen precinct. Based on a method of Rivest, who suggests that auditors select precincts using a method like Percent by Probability rather than selecting a certain sample size as in Exact Percent [11] this algorithm first selects precincts by probability and then does the same for ballots in those precincts. The probability of choosing a given ballot is $p$, where $p \geq 1 - (1 - c)^{\frac{1}{B}}$, and $c$ is the user-specified confidence level. Thus, the probability for a given precinct to be chosen is the probability that at least one of its ballots would be chosen for sampling, or $1 - (1 - p)^v$, where $v$ is the total number of votes in that precinct. The number $k$ of ballots chosen from a selected precinct is determined by a more complicated

distribution, and ballots are chosen for manual auditing by shuffling the list of ballots in that precinct and selecting the first $k$ ballots.

### A.3 Percent By Probability

In this test, all precincts have a user-defined probability of each precinct being selected for manual inspection. The user defines a percentage probability in the range (0, 100], and a random number is generated for every precinct. If the random number associated with that precinct is less than the decimal form of the user-defined percentage, then the precinct is selected for auditing. By probability, this algorithm will select approximately the user-defined percentage of precincts from the race being audited.

### A.4 Exact Percent

Because the probabilities in the above algorithm are evaluated precinct by precinct, the number of precincts selected will vary, and there is a nonzero chance that none at all will be selected. Exact Percent corrects this by guaranteeing that at least the user-defined percentage of precincts is selected for each race. In this algorithm, each precinct is also assigned a random number. Then the precincts are sorted in ascending order by their associated random numbers. If $n$ is the number of precincts to be selected for that race, which is defined as the user-specified percentage times the total number of precincts in the race, rounded up, then we select the $n$ precincts with the smallest random numbers, which always guarantees an output of at least one precinct.

## B Linking Precincts

In these last two algorithms, since precincts are chosen at random depending on the user-entered percentage, it is possible to reduce the costs of manual auditing by minimizing the number of precincts that are chosen for auditing while maintaining the randomness of the selection. In both implementations, each precinct is assigned a certain random number, which is then used for later comparisons. If, in a single audit, more than one race will be audited with either of these two algorithms, then instead of generating a new random number for each precinct in each algorithm, we can reuse the random numbers assigned to precincts that have been evaluated before.

First we take the union of the sets of precincts that are covered by each race. For each precinct in that union, we generate a random number. Then, we audit each race with the one of the two above algorithms (specified by the user) using that same mapping of precincts to random numbers to make selections in every race. Because the random numbers are chosen independently for each precinct, the selection is still random for any given race that uses this dictionary. The Percent by Probability algorithm selects all precincts with random numbers less than or equal to a certain threshold, so precincts that are assigned the lowest numbers and overlap between races will be selected multiple times if more than one race is audited. Exact Percent also selects the required percentage of precincts with the smallest random numbers assigned, so the same precincts will also be chosen multiple times if the same random number mapping is used for all races running this algorithm and those running Percent by Probability.

This algorithm for linking precincts does not maximize the number of precincts audited, because that would require identifying the precincts that overlap across the greatest number of races. That approach would unfairly bias the selection toward those precincts with the greatest number of overlaps. Such a bias would encourage an attacker to target precincts that do not overlap, since they would have less likelihood to be chosen for auditing, undermining the auditing process. Here, however, precincts are all given the same probability of being chosen initially, so using the same mapping in the selections for each race only biases results toward precincts that have already been chosen randomly and independent of other identifying features. The linking of precincts is simply a consequence of reusing the mapping.