Security Analysis of Scantegrity, an Electronic Voting System

Jonathan Ganz[1,2] jmganz@ucdavis.edu
Matt Bishop[1] mabishop@ucdavis.edu
Sean Peisert[1,2,3] speisert@ucdavis.edu

[1]University of California, Davis
[2]Lawrence Berkeley National Laboratory
[3]Corporation for Education Network Initiatives in California (CENIC)

June 27, 2016

Abstract

Electronic voting machines are becoming an increasingly popular alternative to paper ballots. With this increase in use we must analyze how well these machines adhere to voting requirements, including those relating to security, privacy, and anonymity. This paper includes analysis of the security and auditing mechanisms of the open-source electronic voting system Scantegrity. We focus on Scantegrity, not to single it out as a vulnerable system, but because it is a popular system that has actually been used in practice. It may contain design flaws and vulnerabilities that might exist in other systems of similar designs, current or future. Therefore, it is our hope that the vulnerabilities that we bring to light will be considered by current and future designers of electronic voting systems, and that the solutions that we propose will also be considered as possible remediations.

1 Introduction

Security and privacy are significant issues in elections throughout the world. While these issues have long been present in elections that are purely "analog," the rush to systems that involve a digital component of some form has created new types of vulnerabilities. Following the widely publicized study of Diebold Election Systems before the 2004 U.S. presidential election [KSRW04, Cel04], interest in researching and fixing the vulnerabilities of the existing systems skyrocketed. Since then, many different voting systems and schemes designed with security and privacy in mind have been introduced, but most have failed to gain any real traction. One reason is that most of these systems are largely incompatible with the existing infrastructure. That is, in order to implement a lot of these proposed systems, dramatically new equipment has to be designed, bought and integrated, and election procedures often have to be changed considerably as well.

In this paper, we analyze Scantegrity II [CCC$^+$08, CCC$^+$10] (hereafter referred to simply as "Scantegrity"), an end-to-end verifiable voting system using paper ballots, optical scans, and computer-based tallies. We focus on Scantegrity, not to single it out as a vulnerable system, but because Scantegrity may be a perfectly reasonable system from an implementation standpoint for a variety of reasons, including that it might use existing equipment, and, given its use of paper ballots, it is compatible with many existing election procedures. We also note that it is an open-source software system, thereby enabling easier analysis and verification. However, before such a system can be deployed, it has to be tested and analyzed for security flaws. Of particular interest to us in our analysis is safe-guarding the individual voters' rights throughout the election process.

We intend in this paper to focus primarily on a *technical* analysis of the Scantegrity system. Given that all voting systems function in the context of a set of laws and procedures, ranging from ballot design [Nor06] to poll worker training [HBS$^+$12], those must be taken into account as well for a full analysis. Indeed, election procedures can both solve and create vulnerabilities [SEC$^+$10]. However, given that election procedures vary widely in the U.S. and around the world, we mention only some of the more common, high-level election procedures and voter assumptions that would have to change in order for the use of an end-to-end cryptographic voting system such as Scantegrity to be possible.

In the rest of this paper, we first evaluate what information can be determined from the audit logs and attempt to determine whether the voters' privacy and anonymity is preserved throughout the voting process. We then analyze the usefulness of these audit logs in detecting fraud. This balance between privacy and usefulness is a type of "data sanitization" problem: the more information provided in the audit logs, the easier it is to detect malicious activity, but this is accompanied by a greater ease in determining how individuals have voted. We then analyze the security of the ballot definition files that these voting systems rely on. We attempt to generate malformed ballots that may cast votes incorrectly or neglect candidates, preventing the voter from voting as desired, and evaluate Scantegrity's ability to detect the fraudulent ballot definition files. Finally, we propose a set of possible solutions to the vulnerabilities that we found in the system.

2 Voting System Requirements

Election procedures in the United States are governed by a stringent, albeit varied set of rules and requirements. For an election to function correctly and maintain the trust of those who use it to elect their representatives, there are a number of important considerations that must be taken into account. One of the first goals for any election is accuracy [Yee07]. Each registered voter should be presented with one and only one correct and complete ballot for the election and precinct in which he or she intends to vote. After voting, it is also critical that the voter's recorded vote match his or her intent; the voting system should record only and exactly those votes cast by the voter and nothing else.

In addition to requiring that elections have votes counted accurately, they should also be held fairly: each and every voter should be able to cast their vote exactly as he or she intends, without fear of reprisal by either the candidates or an invested third party [Yee07]. In order to ensure fairness, ballots should be anonymous and prevent anyone from verifying how another individual voted. Not only does this prevent voter intimidation, it prevents people from selling votes. Once a vote has been cast, it is also important that the voter's ballot be counted properly, without any bias toward how he or she voted.

These two basic requirements instill in the public the sense of trust necessary to make a democratic election possible in the first place. Therefore, any system that is to be trusted by the electorate has to be able to prove that it works exactly as intended.

3 Overview of Scantegrity

Scantegrity [CCC$^+$08] is an attempt at creating a single voting system that guarantees end-to-end verification. The system is designed to allow voters to not only cast their ballots anonymously and securely, but to also allow an individual voter to verify that the system actually recorded his or her vote as he or she intended it to be cast. Furthermore, Scantegrity was designed for use with optical scan voting systems, such as those already widely deployed in the United States and elsewhere. A direct implication of this design choice is that Scantegrity could theoretically be deployed in precincts nationwide without requiring a major equipment overhaul. The only changes that would have to be made in deploying Scantegrity would be a ballot redesign.

The ballots that the Scantegrity system uses are essentially identical to those used in traditional optical scan elections, except that there is a uniquely mapped code letter next to each choice on the ballot (see appendix 8.1 for an example ballot). Each voter receives a ballot that has a unique ordering of these code letters next to each and every choice that appears on the ballot. Because the code letters are randomized on the ballot, a code letter does not necessarily correspond to a particular vote. This allows for verification by each voter, without revealing how the individual voted. Because Scantegrity's verification process is designed as a zero-knowledge proof, a third party cannot determine how someone else voted. As the voter marks his or her choices on the ballot, he or she may record the letters associated with their vote. This is not mandatory, and may be skipped if the user does not wish to verify the proper casting of their ballot after the election. These code letters, in conjunction with the serial number on the ballot receipt, then allow the user to verify that their ballot was cast correctly.

Scantegrity, like most end-to-end verification systems, produces a receipt of sorts to allow the voter to check on the integrity of the election being held after they have voted. Since this receipt is designed in such a way as to not reveal the identity of the voter or the choices made by the voter during the election, the receipt can be freely given to anyone. With this receipt in hand, an individual can then access the publicly posted election results to ensure that his or her vote is included and that the signature associated with the public results matches up with the signature that the voter has recorded.

The process of running an election with Scantegrity can be separated into three separate steps: 1) the pre-election configuration, 2) the election, and 3) the post-election analysis. During the pre-election configuration, election officials define the ballot template to be used in their precinct and configure Scantegrity to generate the appropriate number of ballots, with random verification code letters next to each choice. During the election, registered voters obtain their ballot, vote on it, optionally record their ballot serial number and the random letters associated with their votes, and cast the ballot. After the election closes, the post-election analysis occurs, in which Scantegrity tallies the votes for each ballot and creates a relation between the ballot serial number and the anonymous verification code associated with the voted options. Once the post-election analysis step completes, voters are free to verify that their ballot was recorded as cast. Scantegrity can also run an auditing process that verifies the integrity of its log files in an attempt to detect any unauthorized modifications that could affect election results.

4 Vulnerabilities in Scantegrity

In our threat model, we assume that the system has been configured as instructed in the provided documentation [Pop08, PH10], and that only election officials have authorized access to the computers storing the results of the election. We assume that poll workers do not have the password used by Scantegrity, nor do they have access to read or modify the configuration files and raw ballots used by Scantegrity. Therefore, poll workers are not considered election officials in this analysis. Furthermore, we assume that any voter may have been approached by a third party, attempting to bias the outcome of a particular race. Voters may also be interested in determining how others have voted or in changing how others have voted. Given these assumptions, the following security vulnerabilities exist with the Scantegrity system.

4.1 Access Control Issues

System Configuration A serious vulnerability is that the system architects do not provide any concrete implementation instructions. There is no information to be found on how a polling entity should go about setting up the Scantegrity system so that it only presents the public with need-to-know information. Without clear instructions for configuring a system to run Scantegrity, the system might present information that allows the average user to determine how others have voted. The Scantegrity system maintains all information relating to the entire voting process in a single directory; within this one directory are a few more directories and various files. Some of the information presented in this top level directory is very sensitive, and access to it should be tightly controlled. In its current form, any election official running the Scantegrity system can easily gain access to any and all files relating to the election. For the purposes of this analysis we have assumed that voters do not have access to these directories. This means that even though the data in this top level directory is not freely available to the voting public, it is available to the election officials at large.

Replay An election official who has access to the files generated by Scantegrity during the pre-election configuration and who also knows the username and password used for that precinct can later determine how each specific ballot is cast using the public audit logs. `MeetingThreeOut.xml` (found in appendix 8.2), an audit log file generated during the post-election analysis, stores how each ballot is cast in a random order and with anonymous identification numbers instead of with the actual ballot serial number. But `SerialMap.xml` (found in appendix 8.3), another audit log file available to the public, is used along with Scantegrity's username and password to generate the anonymous identification numbers and the order in which the ballots are listed in `MeetingThreeOut.xml`. Using the Scantegrity files obtained before the election, an election official can reproduce the election results by reordering the association between ballot serial numbers and votes cast until a `MeetingThreeOut.xml` file is generated that matches the `MeetingThreeOut.xml` file published from the official election. When this occurs, the election official will have the right votes for each ballot, and will know how each ballot serial number was cast. This is a flaw in Scantegrity's design because it produces a mechanism for verifying anyone's vote independently, and can be used for voter intimidation or for buying votes.

Because this attack requires permuting the ballots until the correct order is found, it has a computational complexity of O(n!). With current technology, this running time should prevent attackers from de-anonymizing ballots within a voter's lifetime, so long as enough ballots are cast.

For example, if a precinct casts thirty ballots, an attacker would need to run trillions of modern processors simultaneously to determine how each ballot was cast within the liftetime of voters. However, if more voting machines are introduced in precincts where populations are not increasing, the number of ballots to permute for each machine will decrease, making such an attack faster to execute. At some value of n, it becomes easier to crack the key used to secure the cast ballots. Casting the unvoted ballots in Scantegrity can increase n, guaranteeing a certain level of security, but the auditability of the election would suffer from such actions. This would cause inconsistencies between the number of ballots cast and the number of voters that entered each precinct. Due to the privacy requirements of elections, an unvoted ballot will be indistinguishable from a ballot that was voted on. Once again, we observe a trade-off between privacy and verifiability.

Carrying out this attack could possibly be accomplished using a brute-force approach if voter turnout is extremely low for a precinct. If a third-party was attempting to buy votes and wanted to verify that a compromised voter did indeed vote as asked, it should not matter if this verification of the vote happens in a matter of minutes or in a matter of years—presumably, if the third-party wanted to harm a voter for an incorrect vote, they would do so regardless of the time lapse between the actual casting of the ballot and the third-party's discovery of how the ballot was cast, as long as the voter is still alive.

Logging Configuration A Scantegrity implementation can be configured in such a way that ballots are interpreted incorrectly, enabling precinct-wide electoral fraud. The file `geometry.xml` (found in appendix 8.4), used by Scantegrity to determine how to read the ballots, can be modified to read the ballots in a way that is not intended, and the audit logs do not detect such tampering. We were able to modify `geometry.xml` so that a vote of `Yes` on a contest would be recorded as a vote of `No` and a vote of `No` would be recorded as a vote of `Yes`. This is accomplished by switching the values stored in *id* for a particular contest. For instance, to reverse how the votes are recorded for the second contest, *id* on line 21 should be assigned the value 1 and *id* on line 23 should be assigned the value 0. We were unable to assign both *id*'s the same value for a single contest because the program that encodes the ballot images throws an exception and does not encode the ballots if the `geometry.xml` file is malformed in this manner. This vulnerability is particularly serious because a biased election official can exploit it to reverse election results in precincts that would vote against the official's desired outcome. This vulnerability currently does not require modification of the Scantegrity executables, it is performed in the time between the pre-election configuration and the election, and it currently is not detected by the auditing systems.

4.2 Cryptographic Vulnerability

Another vulnerability is that the information that is meant to be released to the public is not presented in a cryptographically secure fashion (see `SerialMap.xml` in the appendix). The pseudo-random anonymized ballot identifiers have observable patterns which may be vulnerable to a cryptographic attack. More specifically, a correspondence may be derived between the ballot serial numbers and the anonymized serial numbers. In analysing the output of mock elections that we ran, we noticed that the serial numbers and the anonymized serial numbers always increased in value. The developers incremented the serial numbers with pseudo-random values to produce nonces for the anonymized serial numbers. An attack focusing on this relationship can be explored in future work. We found that the number of ballots requested determines how many anonymized identifier numbers are generated in the `SerialMap.xml` file. Only half this number can be used as virtual ballots. This seems to be done so that the `SerialMap.xml` file cannot be constrained to all possible serial numbers, increasing the correspondence between anonymized identifier and ballot serial number. The pseudo-random assignment of anonymized identifiers to ballot serial numbers and the pseudo-random ordering of these anonymized identifiers in `MeetingThreeOut.xml` has also been observed to lack variation upon multiple executions of the post-election analysis, given the same files generated by the pre-election configuration. This cryptographic vulnerability, in conjunction with the previously mentioned access control vulnerability, is what allows an election official to determine how each ballot was cast.

4.3 Coercion To Abstain From Voting

Another attack that has been considered is one in which a third-party coerces, intimidates, or pays individuals, not to vote for a particular candidate, but to refrain from voting on specific contests altogether. When an individual votes for a candidate, the verification code is recorded and then published for anyone to see. If the voter does not vote on a specific contest, no verification code will be recorded for that contest. When the third-party wishes to confirm that an individual did not vote on a contest, the third-party simply enters the individual's verification number and will be presented with the associated anonymous code letters. If the third-party sees a code letter corresponding to the specific contest, the third-party will know that the individual did not act as instructed. Like the `geometry.xml` exploit, this attack is useful when the attacker knows that individuals intend to vote against their desires.

4.4 Possible Verification Vulnerability

Scantegrity is designed as a verifiable voting solution that can fit into many existing processes more easily than many other proposed end-to-end verifiable solutions. However, security vulnerabilties arise when considering this verification process. This process is designed to convince the voter that his or her vote was recorded properly, without revealing to anyone how the voter's ballot was cast.

Scantegrity relies on random letters associated with each contest, combined with a unique ballot serial number to provide this secure verification. The user records both the ballot serial number and the sequence of random letters associated with their vote. After their ballot is recorded, the voter can verify that their vote was recorded properly by entering their serial number into the verification site and comparing the letter sequence there with the one they recorded. If the sequence matches, then it is assumed that the vote was recorded properly. If the sequence does not match, then it can be assumed that the vote was not recorded properly.

There are two obvious methods of exploiting this design. The first method is to attack the Scantegrity software, causing it to properly record the random letter associated with a user's vote, but to ignore the actual vote, recording instead a random vote or a vote for a specific choice. When the user goes to verify the vote, the correct sequence of letters will appear, causing the voter to believe that their vote was recorded properly when it was not. The second method to exploit this design is to attack the web server performing the verification service. If the web server is compromised, the verification method could instead display random letters that are not associated with the ballot ID entered. This will generally not match the sequence of letters that the user recorded, causing voters to believe that their vote was not recorded properly, when the actual ballots recorded had not been tampered with. This will undermine the credibility of the election results.

In the first case, the votes are actually tampered with. In the second case, the perception that tampering has occurred is achieved. According to the documentation provided with Scantegrity, it is the responsibility of other groups to design and implement the web verification service. Therefore, the second exploit is not within the threat model of the Scantegrity designers; but if credibility of the election results is an issue, that attack is still important to consider.

4.5 k-Anonymity Vulnerability

There is potential for a k-anonymity vulnerability [Swe02] using covert channels within the ballot itself [PBY09, Wag10]. For example, if a third-party buys someone's vote for a particular race and wants to confirm that they voted as desired, the third-party can tell them to vote in a specific pattern that the third-party can recognize. This pattern should be unique and unexpected of a normal voter. For instance, the third-party can tell voters to vote such that their verification codes produce a specific sequence of letters, except on the race that matters, in which they vote for the candidate as instructed by the third party. Covert channels are difficult to prevent: attempting to identify the use of covert channels on election ballots may be viewed as a violation of voter privacy, and nullifying ballots that implement a covert channel may be viewed as a violation of the individual's right to cast their ballot as intended. k-anonymity is of concern in this case because it is expected there will be some number k of voters who have cast their ballot with a covert channel implemented. Therefore, k ballots should be identical, so a third-party can determine how many voters did not cast their ballots as determined. Scantegrity would have to sanitize the audit logs so that a single ballot cannot be identified as unique from any $n > k$ other ballots. In our analysis of Scantegrity, we have not identified an example of a k-anonymity vulnerability that does not rely on other vulnerabilities.

5 Proposed Solutions

A variety of solutions have been discussed for validating the security of electronic voting systems by providing auditability while maintaining privacy and anonymity [BEO$^+$12, CW08, CMJ09, BPH$^+$09, SW07]. Those solutions, while useful in many contexts, are unlikely to work directly with Scantegrity, so we propose a variety of alternative solutions, including both procedural and technical approaches.

Some of the access control issues can be resolved by preventing any one election official from being able to view all ballots. This will make it difficult for an individual election official to determine how people voted, because the probability that the official has access to the ballots that the official wishes to verify decreases as the number of ballots accessible to an individual decreases. Scantegrity should also explicitly document the access control structure required to operate in a secure state. The Scantegrity software can either implement this access control structure, or it can verify the structure and operate only if the access controls are secure. In order to fix some of the cryptographic security vulnerabilities, we suggest that election officials randomize the audit log contents using a process external to Scantegrity. A keyed randomizer is needed given that the randomizing algorithm will surely have to be made public. This external randomization process can be automated after Scantegrity completes execution, and should be performed before any audit log data is released. The randomness used by Scantegrity should also be re-evaluated, as it has led to some of the previously mentioned security vulnerabilities. The audit logs may also be sanitized to a greater extent: instead of revealing how each individual ballot is cast, the audit logs can present a summary of how groups of ballots were cast, where each group can be a set of sequential anonymized identifiers.

6 Future Work

Given a prolonged period to perform analysis on the Scantegrity system, we would like to build an automated system capable of implementing our replay attack in order to determine how an individual may have voted, based on the information presented at the outcome of the election. Such work could possibly be accomplished using a brute-force approach depending on the voter turnout for a precinct of average size. Time is not necessarily an issue in solving such a problem, hence a brute-force attack may not be all together unrealistic. More specifically, if, say, a third-party was attempting to buy votes and wanted to verify that a compromised voter did indeed vote as asked, it should not matter if this verification of the vote happens in a matter of minutes or in a matter of days—presumably, if the third-party wanted to harm a voter for an incorrect vote, they would do so regardless of the time lapse between the actual casting of the ballot and the third-party's discovery of how the ballot was cast. We are also interested in determining if an individual with lower permissions than an election official can perform a similar replay attack to determine how individual ballots were cast. This would require more extensive analysis of how Scantegrity implements its cryptographic protocols to randomize and anonymize ballots.

7 Conclusions

Scantegrity represents a promising attempt at implementing a verifiable voting system. The security vulnerabilities identified in this paper have solutions that can be implemented within a reasonable time frame, although the last vulnerability, regarding the absence of verification codes, may require a significant redesign of how verification is performed. The most significant vulnerability is the fact that the vote data released to the public may not always be cryptographically secure enough, and that it could be exploited by a third-party interested in rigging an election that consists of at least a single election official. A simple solution that may prevent voter intimidation to refrain from voting is to include a code letter corresponding to the absence of a vote in each contest. If an individual declines to vote on a contest, the verification code corresponding to "Decline to Vote" will be recorded. This fail-safe default will prevent a third-party from being able to verify that a user refrained from voting as requested.

As we stated in the introduction to this paper, while we certainly hope that our analysis will be of interest to designers and users of Scantegrity, it is our greater hope that, in analyzing the vulnerabilities and proposing possible solutions, designers of other voting systems can leverage the analyses to assist in enforcing security, privacy, and anonymity as well.

Acknowledgements

The authors are grateful to Dr. Stefan Popoveniuc of the Scantegrity project for helping us gain a better understanding of the design of the Scantegrity voting system and the ballot files it uses.

This material is based upon work supported by the National Science Foundation under Awards CNS-0831002, CCF-0905530, CNS-1049738, CCF-0820198, and IIS-0705772, and by the National

Institute for Standards and Technology under Award O-60NANB13D165. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF or NIST.

References

[BEO+12] Patrick Baxter, Anne Edmundson, Keishla Ortiz, Ana Maria Quevedo, Samuel Rodriguez, Cynthia Sturton, and David Wagner. Automated analysis of election audit logs. In *Proceedings of the 2012 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE)*, Bellevue, WA, 2012. USENIX Association.

[BPH+09] Matt Bishop, Sean Peisert, Candice Hoke, Mark Graff, and David Jefferson. E-voting and forensics: Prying open the black box. In *Proceedings of the 2009 Electronic Voting Technology Workshop/Workshop on Trustworthy Computing (EVT/WOTE)*, Montreal, Canada, August 10–11, 2009. USENIX Association.

[CCC+08] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L Rivest, Peter YA Ryan, Emily Shen, and Alan T Sherman. Scantegrity ii: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *Proceedings of the 2008 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT)*, San Jose, CA, 2008.

[CCC+10] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L Rivest, Emily Shen, Alan T Sherman, and Poorvi L. Vora. Scantegrity ii municipal election at takoma park: The first e2e binding governmental election with ballot privacy. In *Proceedings of the 19th USENIX Security Symposium*, Washington, D.C., August 11–13, 2010. USENIX Association.

[Cel04] RABA Innovative Solution Cell. Trusted agent report diebold accuvote-ts voting system. Technical report, RABA Technologies LLC, Columbia, MD, January 2004.

[CMJ09] Paul T Cotton, Andrea L Mascher, and Douglas W Jones. Recommendations for voting system event log contents and semantics. In *NIST Workshop on a Common Data Formats for Electronic Voting Systems*, October 2009.

[CW08] Arel Cordero and David Wagner. Replayable voting machine audit logs. In *Proceedings of the 2008 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT)*, San Jose, CA, 2008.

[HBS+12] Joseph Lorenzo Hall, Emily Barabas, Gregory Shapiro, Coye Cheshire, and Deirdre K Mulligan. Probing the front lines: Pollworker perceptions of security & privacy. In *Proceedings of the 2012 Workshop on Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE)*, Bellevue, WA, 2012. USENIX Association.

[KSRW04] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Daniel S. Wallach. Analysis of an electronic voting system. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 27–40, Oakland, CA, 2004.

[Nor06] Lawrence Norden. The machinery of democracy: Voting system security, accessibility, usability, and cost. Technical report, NYU: The Brennan Center for Justice, 2006.

[PBY09] Sean Peisert, Matt Bishop, and Alec Yasinsac. Vote selling, voter anonymity, and forensic logging of electronic voting machines. In *Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS)*, Jan. 5–8, 2009.

[PH10] Stefan Popoveniuc and Ben Hosp. Explaining the punchscan voting system. In *Towards Trustworthy Elections: New Directions in Electronic Voting*, volume LNCS 6000. Springer, 2010.

[Pop08] Stefan Popoveniuc. Step by step instructions on how to run a scantegrity election on your own computer. Technical report, George Washington University, 2008.

[SEC^{+}10] Borislava I. Simidchieva, Sophie J. Engle, Michael Clifford, Alicia Clay Jones, Sean Peisert, Matt Bishop, Lori A. Clarke, and Leon J. Osterweil. Modeling faults to improve election process robustness. In *Proceedings of the 2010 Electronic Voting Technology Workshop/Workshop on Trustworthy Computing (EVT/WOTE '10)*, Washington, D.C., August 11–13, 2010. USENIX Association.

[SW07] Daniel Sandler and Dan S Wallach. Casting votes in the auditorium. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, Boston, MA, 2007.

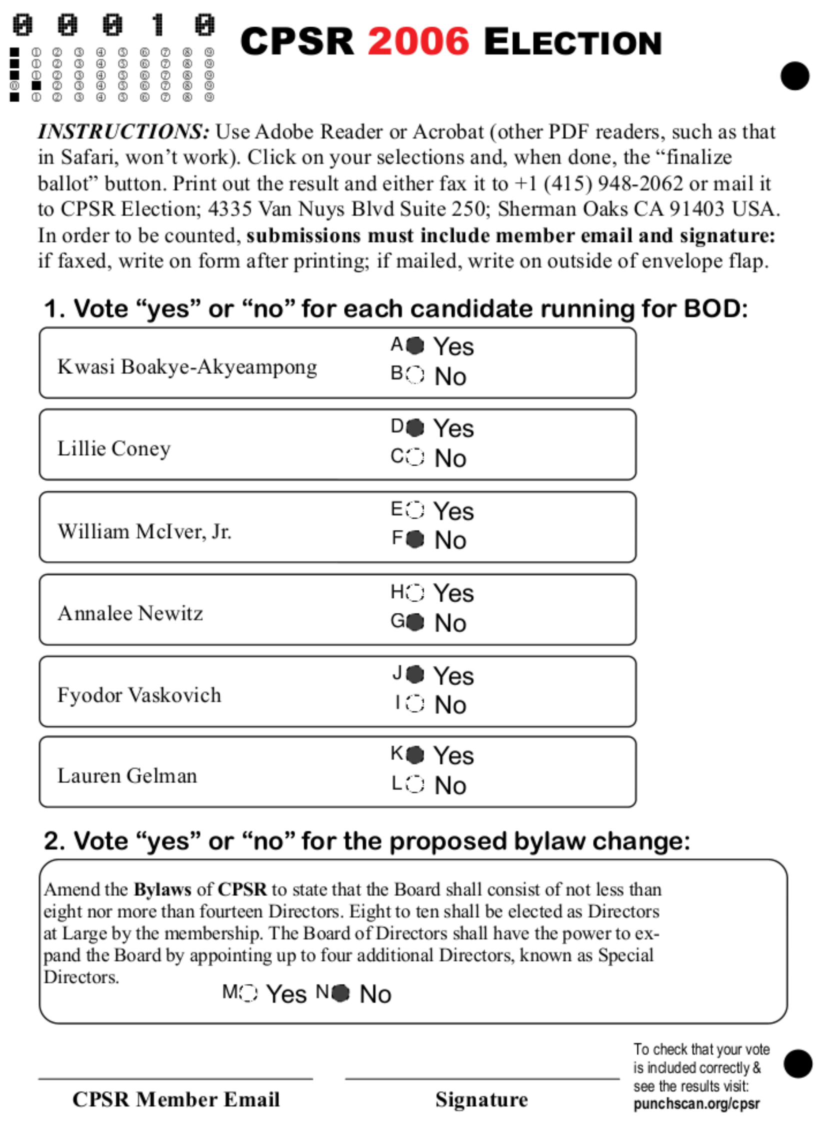
[Swe02] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[Wag10] David Wagner. Voting systems audit log study. *University of California, Berkeley*, 2010.

[Yee07] Ka-Ping Yee. *Building Reliable Voting Machine Software*. PhD thesis, University of California, Berkeley, 2007.

8 Appendix

8.1 Sample Ballot:



0 0 0 1 0

CPSR 2006 ELECTION

INSTRUCTIONS: Use Adobe Reader or Acrobat (other PDF readers, such as that in Safari, won't work). Click on your selections and, when done, the "finalize ballot" button. Print out the result and either fax it to +1 (415) 948-2062 or mail it to CPSR Election; 4335 Van Nuys Blvd Suite 250; Sherman Oaks CA 91403 USA. In order to be counted, **submissions must include member email and signature:** if faxed, write on form after printing; if mailed, write on outside of envelope flap.

1. Vote "yes" or "no" for each candidate running for BOD:

Kwasi Boakye-Akyeampong	[✓ A Yes] B No
Lillie Coney	[✓ D Yes] C No
William McIver, Jr.	E Yes [✓ F No]
Annalee Newitz	H Yes [✓ G No]
Fyodor Vaskovich	[✓ J Yes] I No
Lauren Gelman	[✓ K Yes] L No

2. Vote "yes" or "no" for the proposed bylaw change:

Amend the **Bylaws** of **CPSR** to state that the Board shall consist of not less than eight nor more than fourteen Directors. Eight to ten shall be elected as Directors at Large by the membership. The Board of Directors shall have the power to expand the Board by appointing up to four additional Directors, known as Special Directors.

M Yes [✓ N No]

CPSR Member Email **Signature**

To check that your vote is included correctly & see the results visit: **punchscan.org/cpsr**

Figure 1: A sample ballot generated by Scantegrity. Contests have been voted on. Note that the contests for Lillie Coney, Annalee Newitz, and Fyodor Vaskovich have code letters that are not alphabetically ordered.

8.2 MeetingThreeOut.xml:

```
<xml>
  <database>
    <print>
      <row id="4" p2="0 1 1 0 0 1 0 1 0 1 1 0 1 0" s2="T64LfiLGslSndV5EIgR5MQ=="/>
      <row id="6" p2="0 1 0 1 1 0 0 1 1 0 0 1 1 0" s2="FXp+yvJpC7oMegEjY9tKrg=="/>
      <row id="7" p2="1 0 1 0 1 0 0 1 0 1 1 0 0 1" s2="hu6o4geCQYHB+iu9JGk9tg=="/>
      <row id="8" p2="1 0 1 0 1 0 0 1 0 1 1 0 1 0" s2="Mym8IVlWy/ZywM1LRkaRlA=="/>
      <row id="12" p2="0 1 0 1 0 1 0 1 1 0 1 0 1 0" s2="Smti+bnjvv9mQimgCGUC6g=="
          />
    </print>
    <partition id="0">
      <decrypt>
        <instance id="0">
          <row id="11" d3="1 1 1 1 1 0 1"/>
          <row id="14" d3="0 1 -1 0 0 0 1"/>
          <row id="19" d3="0 0 1 0 0 1 0"/>
          <row id="82" d3="1 0 1 1 1 1 0"/>
          <row id="92" d3="1 0 -1 -1 0 0 1"/>
        </instance>
        <instance id="1">
          <row id="41" d3="1 0 0 1 0 0 1"/>
          <row id="51" d3="1 0 0 0 0 0 0"/>
          <row id="77" d3="1 1 0 0 1 0 0"/>
          <row id="89" d3="1 1 -1 0 0 1 1"/>
          <row id="94" d3="1 1 -1 -1 0 1 0"/>
        </instance>
        <instance id="2">
          <row id="24" d3="1 0 -1 0 1 1 0"/>
          <row id="25" d3="0 1 1 1 1 1 0"/>
          <row id="59" d3="1 1 0 1 0 0 0"/>
          <row id="66" d3="0 0 0 0 0 1 0"/>
          <row id="92" d3="1 0 -1 -1 0 1 1"/>
        </instance>
      </decrypt>
      <results>
        <row id="25" r="1 0 0 1 0 1 0"/>
        <row id="45" r="1 1 1 1 1 1 1"/>
        <row id="54" r="0 0 0 0 0 0 0"/>
        <row id="64" r="0 1 -1 -1 1 0 0"/>
        <row id="67" r="0 1 -1 0 1 0 1"/>
      </results>
    </partition>
  </database>
</xml>
```

8.3 SerialMap.xml:

```xml
<xml>
  <print>
    <row id="0" serial="0"/>
    <row id="1" serial="1"/>
    <row id="4" serial="2"/>
    <row id="6" serial="3"/>
    <row id="7" serial="4"/>
    <row id="8" serial="5"/>
    <row id="12" serial="6"/>
    <row id="15" serial="7"/>
    <row id="21" serial="8"/>
    <row id="22" serial="9"/>
    <row id="24" serial="10"/>
    <row id="25" serial="11"/>
    <row id="26" serial="12"/>
    <row id="29" serial="13"/>
    <row id="32" serial="14"/>
    <row id="35" serial="15"/>
    <row id="38" serial="16"/>
    <row id="39" serial="17"/>
    <row id="41" serial="18"/>
    <row id="42" serial="19"/>
    <row id="46" serial="20"/>
    <row id="47" serial="21"/>
    <row id="48" serial="22"/>
    <row id="49" serial="23"/>
    <row id="50" serial="24"/>
    <row id="51" serial="25"/>
    <row id="54" serial="26"/>
    <row id="55" serial="27"/>
    <row id="59" serial="28"/>
    <row id="60" serial="29"/>
    <row id="61" serial="30"/>
    <row id="62" serial="31"/>
    <row id="67" serial="32"/>
    <row id="68" serial="33"/>
    <row id="70" serial="34"/>
    <row id="71" serial="35"/>
    <row id="73" serial="36"/>
    <row id="74" serial="37"/>
    <row id="75" serial="38"/>
    <row id="77" serial="39"/>
    <row id="82" serial="40"/>
    <row id="83" serial="41"/>
    <row id="84" serial="42"/>
    <row id="85" serial="43"/>
    <row id="86" serial="44"/>
    <row id="88" serial="45"/>
    <row id="90" serial="47"/>
    <row id="97" serial="48"/>
    <row id="99" serial="49"/>
  </print>
</xml>
```

8.4 geometry.xml:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<ballot heigth="11.000" holeDiameter="0.000" width="8.500">
    <alignments>
        <alignment x="7.560" y="1.037"/>
        <alignment x="7.587" y="9.610"/>
    </alignments>
    <contests>
        <contest id="0">
            <top>
                <row id="0">
                    <candidate fromx="4.187" fromy="3.300" id="0"
                        tox="4.340" toy="3.453" x="4.263" y="3.377"/>
                    <candidate fromx="4.187" fromy="3.553" id="1"
                        tox="4.340" toy="3.707" x="4.263" y="3.630"/>
                </row>
            </top>
        </contest>
        <contest id="1">
            <top>
                <row id="0">
                    <candidate fromx="4.187" fromy="4.013" id="0"
                        tox="4.340" toy="4.167" x="4.263" y="4.090"/>
                    <candidate fromx="4.187" fromy="4.267" id="1"
                        tox="4.340" toy="4.420" x="4.263" y="4.343"/>
                </row>
            </top>
        </contest>
        <contest id="2">
            <top>
                <row id="0">
                    <candidate fromx="4.187" fromy="4.727" id="0"
                        tox="4.340" toy="4.880" x="4.263" y="4.803"/>
                    <candidate fromx="4.187" fromy="4.980" id="1"
                        tox="4.340" toy="5.133" x="4.263" y="5.057"/>
                </row>
            </top>
        </contest>
        <contest id="3">
            <top>
                <row id="0">
                    <candidate fromx="4.187" fromy="5.447" id="0"
                        tox="4.340" toy="5.600" x="4.263" y="5.523"/>
                    <candidate fromx="4.187" fromy="5.700" id="1"
                        tox="4.340" toy="5.853" x="4.263" y="5.777"/>
                </row>
            </top>
        </contest>
        <contest id="4">
            <top>
                <row id="0">
                    <candidate fromx="4.187" fromy="6.153" id="0"
                        tox="4.340" toy="6.307" x="4.263" y="6.230"/>
                    <candidate fromx="4.187" fromy="6.407" id="1"
                        tox="4.340" toy="6.560" x="4.263" y="6.483"/>
                </row>
            </top>
        </contest>
        <contest id="5">
            <top>
                <row id="0">
                    <candidate fromx="4.187" fromy="6.853" id="0"
                        tox="4.340" toy="7.007" x="4.263" y="6.930"/>
                    <candidate fromx="4.187" fromy="7.107" id="1"
                        tox="4.340" toy="7.260" x="4.263" y="7.183"/>
                </row>
            </top>
        </contest>
        <contest id="6">
            <top>
                <row id="0">
                    <candidate fromx="2.740" fromy="8.920" id="0"
                        tox="2.893" toy="9.073" x="2.817" y="8.997"/>
                    <candidate fromx="3.540" fromy="8.920" id="1"
                        tox="3.693" toy="9.073" x="3.617" y="8.997"/>
                </row>
            </top>
        </contest>
    </contests>
    <serial>
        <digit id="0" type="counter">
            <top fromx="3.860" fromy="0.613" tox="4.053" toy="0.847"
                x="3.957" y="0.730"/>
        </digit>
        <digit id="1" type="counter">
            <top fromx="4.087" fromy="0.613" tox="4.273" toy="0.853"
                x="4.180" y="0.733"/>
        </digit>
        <digit id="2" type="counter">
            <top fromx="4.313" fromy="0.613" tox="4.500" toy="0.853"
                x="4.407" y="0.733"/>
        </digit>
        <digit id="3" type="counter">
            <top fromx="4.533" fromy="0.613" tox="4.727" toy="0.853"
                x="4.630" y="0.733"/>
        </digit>
        <bulleted fromx="0.700" fromy="0.787" tox="2.580" toy="1.267"
            x="1.640" y="1.027"/>
    </serial>
</ballot>
```