Casting Votes in the Auditorium

Daniel Sandler
dsandler@cs.rice.edu

Dan S. Wallach
dwallach@cs.rice.edu

Rice University

Abstract

In elections employing electronic voting machines, we have observed that poor procedures, equipment failures, and honest mistakes pose a real threat to the accuracy of the final tally. The event logs kept by these machines can give auditors clues as to the causes of anomalies and inconsistencies; however, each voting machine is trusted to keep its own audit and ballot data, making the record unreliable. If a machine is damaged, accidentally erased, or otherwise compromised during the election, we have no way to detect tampering or loss of auditing records and cast votes.

We see a need for voting systems in which event logs can serve as robust forensic documents, describing a provable timeline of events leading up to and transpiring on election day. To this end, we propose an auditing infrastructure that draws on ideas from distributed systems and secure logging to provide a verifiable, global picture of critical election-day events, one which can survive individual machine malfunction or malice. Our system, the Auditorium, *joins the voting machines in a polling place together in a private broadcast network in which all election events are logged redundantly by every machine. Each event is irrevocably tied to the originating machine by a digital signature, and to earlier events from other machines via hash chaining.*

In this paper we describe in detail how to conduct an election in the Auditorium. We demonstrate our system's robustness to benign failures and malicious attacks, resulting in a believable audit trail and vote count, with acceptable overhead for a network the size of a polling place.

1 Introduction

The ongoing debate over electronic voting systems focuses, by and large, on the trustworthiness of their software implementation. Could an insider build a voting machine which appears correct and passes certification, but still violates the integrity of the vote or the anonymity of the voter? Could an outsider with knowledge of an exploitable flaw in an otherwise honest machine do the same? Certainly the answer to these sorts of questions is "yes," and recent research [22, 29, 41, 21, 11, 18, 19] has rightly investigated the dangers posed by faulty or malicious voting software.

In this work we consider a different facet of the election correctness problem. Leaving aside the issue of voting machine *correctness*, we instead investigate the *auditability* of elections conducted with electronic voting systems. When the results of an election are in doubt, the usual course of action is to perform a recount. When so-called direct recording electronic (DRE) voting machines are in use, the data surveyed during such a recount is purely electronic, and hence, fundamentally mutable. Any party in possession of the machine or its flash memory cards or the tabulation system might be able to alter or destroy votes. What does it mean to recount votes whose provenance cannot be proven?

The problem extends beyond ballots. It is common for DRE machines to keep an an *event log* to support *post facto* analysis when the correct operation of the machines comes into question. These logs, which record and timestamp interesting events such as "election started" or "ballot cast," can provide critical clues to the events of election day, especially when the vote tally is unusual or inconclusive. That is, of course, unless they've been tampered with, in which case they prove nothing at all.

We see a need for voting systems in which event logs can serve as robust forensic documents, describing a provable timeline of events leading up to and transpiring on election day. To this end, we propose an auditing infrastructure that draws on ideas from distributed systems and secure logging to provide a verifiable, global picture of critical election-day events, one which can survive individual machine malfunction or malice. Voting machines in our system are joined together on a broadcast

network, entirely separate from the Internet, in which all election events are logged redundantly by every machine. Each event is irrevocably tied to the machine it came from by a digital signature, and to previously-announced events from other machines via hash chaining.

The result is a resilient, tamper-evident timeline of events that can be believed during an audit. Because all auditable events are broadcast, we call this secure shared timeline infrastructure the *Auditorium.* Much like the papal conclave [28, 34]—in which the election of the Pope is conducted in open view of all electors—events in the Auditorium are observed by all voting machines on the network, even though the contents of individual votes are sealed. After an election is over, the Auditorium records from all participating machines can be used to identify and correct procedural and technical problems that cannot be detected in conventional paper audit trails.

While it is clearly necessary to improve the software engineering standards to which DRE systems are held, even a perfectly bug-free and trustworthy DRE software stack is susceptible to operator error and hardware failure. The Auditorium is thus an essential complement to other secure-voting techniques. We are in the process of developing the Auditorium as a part of *VoteBox,* our project to develop a voting machine software platform for both security and human factors research.

In the next section of this paper, we describe the experiences and problems that have motivated our thinking and prompted the design for Auditorium, which is the focus of Section 3. In Section 4, we discuss the security properties of our system in the face of the kinds of everyday failures that call results into question; we also identify and address a few "mega attacks" that challenge any voting system, and consider the unavoidable possibility of malicious or faulty software. We outline our current and future research directions in Section 5. Section 6 concludes.

2 Background

In March 2006 we were fortunate enough to be asked to investigate the results of a primary election in Webb County, Texas. The voters in this jurisdiction were given the option to vote on paper or using the county's new iVotronic touch-screen DRE systems, manufactured by Election Systems & Software (ES&S). The second-place finisher in a local judicial race found that he received a smaller share of the DRE vote than the paper vote, and so contested the electronic election results. Officials therefore impounded the voting machines and allowed us to examine the systems for signs of tampering, malfunction, or other inexplicable irregularities. We submitted our final report [42] to the court.

In the limited time available, we were unable to find any direct evidence of tampering, nor of any malicious or faulty code in the software or firmware of the machines. Due to the malleable nature of electronic storage in paperless DREs (and, as was recently demonstrated [18], the ability of malicious software to erase itself from a DRE after the damage is done), it is fundamentally impossible to be sure that no such tampering or malice occurred.

We did, however, discover anomalous and incomplete information in the *event logs* kept by the iVotronic machines. These logs, stored in a proprietary format on the flash memory inside the voting machines, exist to provide some degree of auditability after the election is over. When the polls close, poll workers copy the contents of the voting machines onto CompactFlash memory cards, which are then transferred to a general-purpose PC running the ES&S tabulation software. An example of its output, given one machine's binary event log, is shown in Figure 1. We examined both the text logs emitted by the tabulation software and the raw binary logs stored on the machines themselves.

2.1 Lost votes

Figure 1 shows something unexpected. While polls opened for the primary election around 7 AM on March 7, 2006, this particular machine was cleared and entered into service at about 3:30 PM that same day. This could be entirely innocuous: perhaps this machine was simply unneeded until the early afternoon, at which point poll workers activated it.

However, the machine might also have been accepting votes since 7 AM like the other machines in that precinct, but was wiped clean in the afternoon. Because the machine is trusted to keep its own audit and vote data—both of which can be erased or otherwise undetectably altered—we cannot be sure that votes were not lost.

There exists a procedure to mitigate against this sort of ambiguous vote record, albeit a fragile one. Official election procedures direct poll workers to print a "zero tape" on each machine before it is entered into service on election day, and a "results tape" once the polls are closed. Each tape reveals (in addition to the election-specific, per-race tallies) the contents of the machine's *protected count*, a monotonic counter inside the machine that is incremented any time a ballot is cast and, according to the manufacturer, cannot be decremented or reset even if the machine is cleared. This is a feature first found on mechanical (lever-based) voting machines.

It should therefore be possible (assuming the counter resists tampering) to compare the difference in the count on the zero and result tapes and the number of ballots recorded on that machine in between. If the numbers are not equal, votes cast on election day were lost, or votes cast on other days are being treated as legitimate, or both.

Unfortunately, the system does not require that these

Votronic	PEB#	Type	Date	Time	Event
5140052	161061	SUP	03/07/2006	15:29:03	01 Terminal clear and test
	160980	SUP	03/07/2006	15:31:15	09 Terminal open
			03/07/2006	15:34:47	13 Print zero tape
			03/07/2006	15:36:36	13 Print zero tape
	160999	SUP	03/07/2006	15:56:50	20 Normal ballot cast
			03/07/2006	16:47:12	20 Normal ballot cast
			03/07/2006	18:07:29	20 Normal ballot cast
			03/07/2006	18:17:03	20 Normal ballot cast
			03/07/2006	18:37:24	22 Super ballot cancel
			03/07/2006	18:41:18	20 Normal ballot cast
			03/07/2006	18:46:23	20 Normal ballot cast
	160980	SUP	03/07/2006	19:07:14	10 Terminal close

Figure 1: An iVotronic event log. The machine in question has the serial number 5140052; several different PEBs (special administrative access tokens) were used over the course of the day. Noteworthy: the machine was cleared and entered into service at about 3:30 PM on election day.

tapes be printed, nor that they be properly stored. In the case of machine 5140052, we were unable to locate a zero tape; the results tape showed a protected count of 12, and we observed 6 votes in the final tally from that machine, so a maximum of 6 votes were lost. It is quite possible that *no* votes were lost, and that the other 6 votes were votes cast at other times for other purposes (e.g., other elections or tests). We cannot be sure, and had the machine been in service for many years, its protected count would be much higher, correspondingly inflating our best upper bound on the number of lost votes.

2.2 Other anomalies and ambiguities

We encountered several machines whose logs attest that votes were cast on those machines on or before March 6, the day *before* the primary election. Some of these machines showed what appeared to be a normal voting pattern, with the exception that every vote was cast on the 6^{th}. Inspection of those machines (an example is shown in Figure 2) revealed that their hardware clocks were off by one day, implying that the votes in question were in fact cast during the election on the 7^{th}. We do not know for sure; anyone with access to the machines prior to the election could have cast these ballots illegitimately.

Other machines (e.g. Figure 3) with votes cast on the wrong day fit a different pattern. In each case, two votes were recorded: one ballot in the Republican primary election and one in the Democrat primary. For each ballot, the particular candidates chosen were the same each time. We learned that this is the profile of a machine under "logic and accuracy" test; election officials would cast a couple of ballots and satisfy themselves that the machines were working. Somehow these test votes were being counted in the official election tally.

We also saw evidence of procedural failures which call into question the accuracy of the vote tally. In Figure 1, the event described as *Super ballot cancel* represents a situation where a "supervisor" (poll worker) had to abort an in-progress voting session. This typically happens when voters "flee," that is, they leave the polling place without completing a ballot. In this event, poll workers are under instruction to *cancel* the incomplete ballot and to *record* on a paper log the reasons for having done so. These paper logs were rarely kept in this particular primary election, so we have no way to confirm the legitimacy of the cancellation. (It is not possible to cancel a vote after it has been successfully cast.)

We conclude from these experiences that electronic voting systems generate a great deal of auditing data that can shed light on irregular results. Because the data for the entire county was available in digital form, we were able to analyze a large amount of election auditing data at great speed, a feat that would have been far more difficult if we relied on paper records kept in each precinct by poll workers. Despite their usefulness, however, we are still able to prove neither the accuracy nor completeness of these event logs.

2.3 Requirements for auditable voting systems

If we wish to design voting systems which survive mistakes and failures to provide an unambiguous result, we must formalize the required properties of such a system. In an *auditable voting system:*

R1 Each machine must be able to account for every vote. Any ballot to be included in the final tally must be legitimate; that is, it must provably have been cast while the polls were open. It must also be possible to prove, by examining the auditing records, that no legitimate votes have been omitted from the tally. This

Votronic	PEB#	Type	Date	Time	Event
5142523	161061	SUP	02/26/2006	19:07:05	01 Terminal clear and test
	161115	SUP	03/06/2006	06:57:23	09 Terminal open
			03/06/2006	07:01:47	13 Print zero tape
			03/06/2006	07:03:41	13 Print zero tape
	161109	SUP	03/06/2006	10:08:26	20 Normal ballot cast
			03/06/2006	12:39:05	20 Normal ballot cast
			03/06/2006	14:49:33	20 Normal ballot cast
			03/06/2006	15:59:22	20 Normal ballot cast
			03/06/2006	18:01:45	20 Normal ballot cast
			03/06/2006	18:10:24	20 Normal ballot cast
			03/06/2006	18:26:52	20 Normal ballot cast
			03/06/2006	18:29:18	20 Normal ballot cast
			03/06/2006	18:39:41	20 Normal ballot cast
			03/06/2006	18:44:24	20 Normal ballot cast
	161115	SUP	03/06/2006	19:29:00	27 Override
			03/06/2006	19:29:00	10 Terminal close

Figure 2: Machine showing the wrong date. These votes appear to be cast on the day *before* the election; when we inspected the machine we found that its hardware clock was off by one day, implying that these are likely to be valid election-day votes.

Votronic	PEB#	Type	Date	Time	Event
5145172	161061	SUP	03/06/2006	15:04:09	01 Terminal clear and test
	161126	SUP	03/06/2006	15:19:34	09 Terminal open
	160973	SUP	03/06/2006	15:26:59	20 Normal ballot cast
			03/06/2006	15:30:39	20 Normal ballot cast
	161126	SUP	03/06/2006	15:38:37	27 Override
			03/06/2006	15:38:37	10 Terminal close

Figure 3: Likely test votes. This machine also shows votes cast on March 6, the day before the election. When we inspected this machine, however, its hardware clock was set to the correct date.

property should extend beyond votes to other important events, such as ballot cancellation.

R2 A machine's audit data and cast ballots must survive that machine's failure. The overall system must defend against the loss of critical election data due to malfunction, loss, destruction, or tampering with individual machines.

Others have designed single-machine tamper-evident ballot storage systems [26, 8], but we go a step further and require that all auditing records kept by voting machines be tamper-evident. Molnar et al. [26] propose a number of other desirable properties for vote storage mechanisms; we discuss these properties in Section 5.3.

We argue that R1 and R2 are sufficient to detect and recover from the procedural errors that we have observed and that can cast doubt upon even legitimate election results. In the following section, we will describe our design for a system which meets these requirements.

3 Auditorium

3.1 How we learned to stop worrying and love the network

Our solution to the problems of resilient, believable audit records revolves around the idea that **we can improve auditability by connecting voting machines to one another.** The general idea of networking the polling place does not originate with us; some electronic voting systems (the Hart InterCivic eSlate, for one) already support the use of a network. However, the elections community has historically been very suspicious of networks, and with good reason: any unjustified increase in the potential attack surface of a voting machine is inexcusable.

A network could make possible two kinds of previously-infeasible attacks: voting machines could be attacked from outside the polling place, and a single compromised voting machine can now attack others from the inside. If a polling place is networked, it must *not* be reachable from the Internet, obviating an outside attack.

Such an "air gap" is already an important part of military computer security practices and is sensible for electronic voting.

The "inside attack" is an interesting case. An attacker needs physical access to only one machine (perhaps the one on which the attacker is voting) in order to install malicious code, which can then spread via the network. We observe that the network is not necessary for this kind of attack. One of the chief features of DRE voting machines is the speed with which they may be tallied; this speed comes from some sort of communication between machines, whether in the form of a network or simply exchanged memory cards. The Diebold AccuVote-TS system uses flash memory cards for this purpose and Feldman et al. [18] found that this card-swapping was an effective way to spread a "voting machine virus." Yasinsac et al. [43] found a similar vulnerability with the ES&S system. In the end, the lack of a network does not guarantee isolation of any faulty or malicious voting machine. We recognize that a networked voting system is necessarily still *more* vulnerable; in the following sections we will show the security properties that justify this additional risk.

3.2 Secure logging

We now build up our design for an auditable voting system from essential building blocks, of which the first is *secure logging.* Our requirement R1 would be satisfied by a voting machine able to produce a tamper-evident record of ballots cast and other pertinent events. A first step is a *secure log,* such as those described by Bellare and Yee [5] and Schneier and Kelsey [35]. Each event in a secure log is encrypted with a key that is thrown away so that, if attackers gain control of a machine, they should be unable to read log messages written in the past (that is, before the attack). Encryption keys are generated deterministically from one another, starting with an initial key that is retained by a trusted party. To read the logs, the sequence of keys can be re-generated, and log entries decrypted, given the initial key.

The inability of untrusted parties to read previous log entries, termed *perfect forward secrecy,* is not necessary for electronic voting, where the pertinent data is a public record. Instead, we need *forward integrity* [5], the property that an attacker may not undetectably remove, add to, or alter auditing records committed before the attack. This can be achieved with *hash chaining.* Each event E_i includes $\mathsf{hash}(E_{i-1})$, the result of a collision-resistant cryptographic hash function applied to the contents of the previous event.

If the contents of E_{i-1} are hard to predict (for example, a random nonce is included), the time at which E_i was committed to the log is now *backward-constrained*: it must succeed the time of E_{i-1}. When event E_{i+1} in turn incorporates the hash of event E_i, E_i is now *forward-constrained* as well. Thus, each event E_i, containing log data d_i, has the form $[d_i, \mathsf{nonce}, \mathsf{hash}(E_{i-1})]$. (Naturally, there must exist a special event E_0 from which the first real event E_1 derives; it can be defined as a well-known arbitrary value, such as a string of zeros of appropriate length.)

3.3 Entangled timelines

Moving beyond the realm of a node's own timeline, we now consider ways to reason about multiple timelines in a distributed system (such as a polling place). The concept of fixing events from foreign, untrusted timelines in the reference frame of local events originates in the logical clocks of Lamport [23]. Maniatis and Baker make this scheme tamper-proof by fusing it with hash-chaining to form what they call "timeline entanglement" [24]. An *entangled timeline* is a secure log which includes, among the links in its hash chain, events from the secure logs of other (possibly untrusted) parties. Alice might, for example, send an event to Bob, who can now mix that information into *his* next event. Now Bob can prove to Alice that his event succeeds hers in time.

Much like including a copy of today's newspaper in a photograph to fix it in time for any skeptical auditor, entangled timelines allow parties to fix the events of others in their own timelines. By following links of hash chains to and from a foreign event in question, a node should be able to eventually reach events in its own timeline that provably precede and succeed it.

3.4 Auditorium: n-way entanglement and replication

With entangled timelines, we are able to satisfy requirement R1 without the need of either a trusted auditor holding a base key or a trusted timestamping service. We have still not met the burden of R2, however; we can detect erasure, but not recover the records.

To this brew of concepts we therefore add insight from peer-to-peer research: in a world where disk and network are cheap and abundant, we have the luxury of widespread replication. While debate continues as to whether these criteria hold in the wider Internet, we argue that it is certainly true for electronic voting: even the low-end computers used in DREs are mostly wasted on the task of voting.

We introduce a simplification of the entangled timeline scheme that is practical for small networks such as a group of voting machines in a polling place. Rather than periodically exchanging a fraction of their events along with hash-chain precedence proofs (as in Maniatis and Baker), our nodes *broadcast* every state change that we will ever want to reason about or recover. This notion forms the kernel of the *Auditorium:* Every event is heard

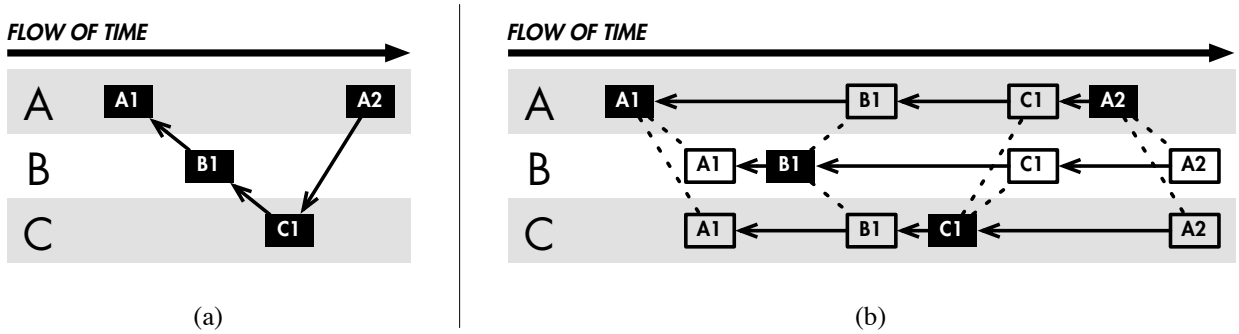

Figure 4: Flow of time in the Auditorium. Participants A, B, and C experience a shared flow of time (a). Solid arrows denote direct temporal precedence; for example, X ← Y illustrates an event Y that incorporates $\mathsf{hash}(X)$, proving that event X must precede event Y in time. The abstraction of a shared timeline is achieved by replicating local events to remote nodes (b); dotted lines represent Auditorium broadcasts.

and recorded by every participant, and each new event is entangled with the last. The resulting abstraction—a single shared flow of time—is illustrated in Figure 4.

Every event is cryptographically signed by its originator to prevent forgery. The use of signatures in conjunction with entanglement prevents later repudiation of events, or even repudiation of the time at which those events occurred. A log entry E_i from node A therefore has the form $A.\mathsf{sign}([d_i, \mathsf{nonce}, \mathsf{hash}(E_{i-1})])$, where $A.\mathsf{sign}(X)$ denotes X signed with A's public key.

As a part of our simplification, we forego the multiple-phase protocol used by Maniatis and Baker to exchange precedence proofs; the result is that the shared timeline may naturally diverge due to asynchrony. For example, Bob and Charlie may both publish events including a hash of Alice's last message. We say that these two events are *contemporaneous;* that is, they cannot be distinguished in time because they directly succeed the same event. If Alice wants to broadcast another event, she has a choice: should her event include the hash of Bob's event or Charlie's? She must choose *both* in order to forward-constrain both events in time. Therefore, her message should *merge the timelines* by including the hash of any prior event not already constrained. We thus allow an event E_i to include the hashes of any number of prior unconstrained events: $E_i = A.\mathsf{sign}([d_i, \mathsf{nonce}, [\mathsf{hash}(E_j), \mathsf{hash}(E_k), \ldots]])$.

3.5 The Auditorium broadcast network

To create the abstraction of a broadcast channel, our implementation uses a fully-connected network of point-to-point TCP sockets. Every participating node in a network of size n is connected to every other, resulting in a complete graph (K_n) of connections. Any new message generated by a participant should be sent on every open connection. To illustrate: if Alice, Bob and Charlie comprise an Auditorium instance and Alice wishes to announce an event to the Auditorium, she sends her message to Bob and Charlie. We further specify that messages should follow a gossip protocol; any node receiving a "new" event (one whose hash it has never before seen) should forward it to every other participant. In our example, Bob and Charlie would each forward the new event to the other; upon receiving what is now an "old" event, no more forwarding would take place.

Such a network, quadratic in the number of open connections, is hardly the only way we could have provided the broadcast abstraction, and is certainly not the most scalable. For the problem of voting, however, n is quite small (see Section 4). We chose this mechanism because it is extremely robust and simple to implement. We need no complex tree-construction algorithms or maintenance operations; every node is simply connected to every other, and shares new messages with them all.

Because new messages are flooded on every link, nodes hear about the same message from every other node in the system. This flurry of seemingly-redundant traffic has the extremely valuable benefit of providing robustness to Byzantine faults in timeline entanglement [24]; a node might attempt to reveal divergent timelines to different neighbors, but in the Auditorium this duplicity will be quickly exposed as conflicting messages are exchanged among their recipients. We discuss ways to apply more sophisticated overlay networks to this problem, accepting some fragility in exchange for scalability, in Section 5.

3.6 Voting in the Auditorium

As we have described it thus far, Auditorium is a general-purpose auditable group communication system, so we must now specify a protocol for holding an auditable election inside the Auditorium. The next several sections detail the voting protocol; the pertinent messages are summarized in Figure 6.

3.6.1 Opening the polls and authorizing ballots

Each polling place will receive a number of voting machines using the Auditorium system (hereafter referred to as VoteBoxes, after our prototype voting machine software), as well as at least one *controller* machine running a special election-control application. Figure 5 illustrates our polling place concept.

The VoteBoxes have identical software configurations; they differ only in their unique identifiers (assigned by the manufacturer) and in their cryptographic keys. Each public key is encapsulated into a certificate signed by a trusted certificate authority (held, for example, by the administrator in charge of elections). The corresponding private key is used for signing messages in the Auditorium protocol. Similar properties hold for controller machines; spares of each may be kept (in storage or active and on the network) to be brought into service at any time.

On election day, the machines are connected to power and to the network and booted. Nodes self-select IP addresses and discover one another using untrusted Ethernet broadcast packets; once discovery is complete, nodes build a fully-connected network such that every pair of nodes maintains an open TCP connection. Nodes joining the network later engage in a similar bootstrapping process. We omit a full discussion of the network bootstrapping algorithm due to space limitations.

When the election is to begin, the controller announces a polls-open message. At this point the polling place is ready to accept votes. Once a voter has signed in with poll workers, the poll workers in turn use the controller's user interface to authorize a particular VoteBox terminal to present the correct ballot for the voter. This is done with an authorized-to-cast message, which includes the ballot definition for that voter's precinct, a nonce, and the ID of the particular VoteBox the voter is to use. (Poll workers will then direct the voter to the correct machine.) All of these messages are broadcast to and recorded by every other VoteBox.

3.6.2 Casting and cancelling ballots

The VoteBox uses the ballot definition inside the authorization message to present a voting interface to the voter. When the voter has finished making selections and presses the final "cast ballot" button in the UI, the VoteBox announces a cast-ballot message, which contains the original authorization nonce for that ballot as well as an *encrypted cast ballot* containing the voter's selections. We discuss the nature of the encryption in Section 3.6.3.

Between the authorized-to-cast and cast-ballot messages we necessarily rely on the correct operation of the VoteBox software to faithfully capture and record the voter's selections. Of course, faulty software might well tamper with or corrupt the vote before it is broadcast. We consider this threat in Section 4.4.

We now add another message to the protocol in order to provide meaningful feedback to the voter that a ballot has been successfully cast. The controller will acknowledge the receipt of a legitimate encrypted cast ballot by announcing a received-ballot message including the appropriate authorization nonce. This has several benefits: it allows the voting machine to display a confirmation message to the user; it de-authorizes the nonce, ensuring it cannot be used again to cast a legitimate ballot; and it tightly constrains the cast ballot in time by entangling the cast-ballot message with one from the controller's timeline.

If the voter flees (that is, decides to leave without casting a ballot), the appropriate procedure may be for a poll worker to cancel that outstanding ballot using the controller. The controller will then announce a cancelled-ballot message, which contains the nonce of the authorization to be revoked. No subsequent cast-ballot message corresponding to the authorization (*viz.*, including its nonce) will be considered legitimate. Likewise, the controller will never send an authorized-to-cast message for a machine with an outstanding authorization; the previous ballot must first be either cast or cancelled before a new one can be authorized.

3.6.3 Ballot storage and encryption

Cast ballots are part of the Auditorium event timeline, and so their order can by definition be reconstructed. This clearly poses a tradeoff with the anonymity of the voter; in general, perfect anonymity tends to stand in the way of auditing. Our system mitigates this particular threat to anonymity by sealing the contents of cast ballots. In our current design, we employ conventional public-key encryption to control access to the plaintext of each ballot. No attacker in possession of the Auditorium logs—perhaps a network eavesdropper, or a malicious party in possession of a VoteBox after election day—can recover any votes, even though all votes are dutifully logged by every VoteBox in the polling place.

Some jurisdictions require each polling place to produce its own legible election results once the polls close, forcing us to allow decryption in the polling place. By encrypting ballots with a public key held by the controller machine, we can enable the controller to decrypt the ballots found in its own log and print the current tally. Alternatively, rather than trusting the controller, each VoteBox could keep counters and announce them on demand; these running tallies could always be verified against the Auditorium logs after the election is over. (Furthermore, if requests for these totals were inappropriately made while the election was ongoing, every VoteBox would record that fact in its logs.)

Precincts may also be required to produce a full paper

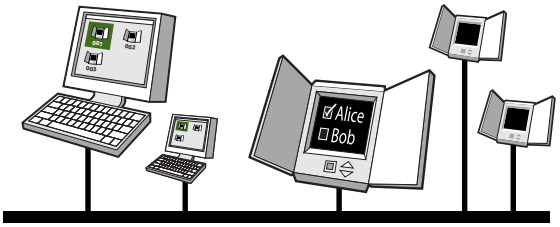

Figure 5: Configuration of machines in a polling place. A number of VoteBoxes are connected to one another and to a controller machine by the Auditorium broadcast network. A redundant controller is also on the network in case of failure; similarly, VoteBoxes may be swapped out at any time.

record of each cast ballot. In this case we must of course be careful not to print those ballots themselves in the order they were found in the log, lest we compromise the anonymity of the votes. A straightforward way to deterministically erase the order of the ballots is to sort them lexicographically [6], although this still requires trust in the controller to perform the sort. Sophisticated cryptographic techniques may be able to shuffle the votes or produce partial vote totals without fully trusting any one machine. We discuss these issues further in Section 5.3.

By contrast, Molnar et al. [26] use a dedicated write-only hardware device to store plaintext ballots in such a way that their order is destroyed (thus preserving anonymity) and that tampering with the finalized vote record is evident. Such a system might be combined with our design, for example, as an attachment to the controller or to the VoteBoxes themselves. Similarly, history-hiding append-only signatures—as proposed recently by Bethencourt et al. [8]—are software-only cryptographic functions that accomplish the same task. If each VoteBox announces the contents of these vote stores at the end of the day, voter anonymity will still be properly preserved.

Unfortunately, the history-hiding properties of these ballot storage techniques are problematic for auditing purposes. Ballots that should not be counted (e.g., because they were cast on the wrong day) are scrubbed of their context and become indistinguishable from legitimate votes. With our secure timeline-based design we therefore choose to *preserve* voting history, allowing election officials to reconstruct the exact voting order if necessary for an audit. (Different jurisdictions have different voter anonymity requirements. Certainly, threshold cryptographic schemes could be used to require the cooperation of multiple, independent parties before the plaintext votes are revealed in order.)

If the veracity of any Auditorium record (machine events or votes) comes into question, auditors can examine the Auditorium logs on the controller and the VoteBoxes involved. Of course, every message is signed by its sender before being broadcast. This signature is logged, along with the rest of the received message. In Section 5.3 we discuss other relevant cryptographic issues and techniques.

3.6.4 Heartbeats; closing the polls

It's possible that a quiescent polling place will go a long time (minutes or even hours) between legitimate election events. This will result in a loss of temporal resolution when later examining the audit logs; to address this, the controller will send periodic heartbeat messages to help fix surrounding events in wall-clock time. VoteBoxes also send heartbeat messages that include additional state, such as the battery level and the protected count. If a VoteBox observes that one of its open TCP connections has been reset, it announces a disappeared message to document the loss of connectivity to the peer at the other end of the socket. All of these messages aid auditors who might be trying to reconstruct the state of the polling place at any given time during the day.

When it is time to close the polls, the poll worker instructs the controller to wait until there are no further outstanding ballots before broadcasting a polls-closed message. This establishes an end to legitimate voting, and places each VoteBox in its post-election shutdown mode.

3.7 VoteBox: a platform for electronic voting research

We are currently in the process of developing VoteBox, a from-scratch Java implementation of an electronic voting system. In addition to allowing us to experiment with novel security infrastructures like the Auditorium, the VoteBox lets us explore other aspects of the voting machine design space.

For example, VoteBox takes advantage of the pre-rendered user interface technique pioneered by Yee et al. [44] as a way to dramatically reduce the amount of trusted code in a voting machine. We take this notion a step further, distributing pre-rendered ballot definition files to voting machines *on demand* using Auditorium messages (see Section 3.6.1).

Because the ballot definition is supplied with each vote authorization, we are free to release the resources associated with it once the voter has cast a ballot. This naturally provides a valuable security property identified by Sastry et al. [33], namely that no state may leak from one voter's session to the next. In fact, in the VoteBox we release all references to the user interface module between voters; the Java virtual machine prevents future execution from ever again reading the contents of that memory.

(**join** *votebox-id public-key-certificate ...*)	Sent by any node as part of the bootstrapping process.
(**polls-open**)	Sent by the controller to start an election.
(**heartbeat** *timestamp status*)	Sent periodically by all nodes to fix events in time.
(**disappeared** *node-id*)	Sent by a node to announce loss of a direct peer link.
(**authorized-to-cast** *votebox-id nonce ballot-definition*)	Sent by the controller to authorize one voting machine to cast one specific ballot.
(**cancelled-ballot** *votebox-id nonce*)	Sent by the controller to retract an authorization.
(**cast-ballot** *nonce encrypted-choices*)	Sent by a VoteBox to cast a completed ballot.
(**received-ballot** *nonce*)	Sent by the controller to acknowledge a cast ballot (ending the authorization).
(**polls-closed**)	Sent by the controller to end an election.

Figure 6: Auditorium messages for electronic voting. Messages are listed in rough order of their likely occurrence in a real election.

Finally, the VoteBox system is intended to support human factors research and experimentation. Studies have been conducted into the usability of currently-deployed voting systems, both electronic [4, 16] and conventional [17, 20], but in order to improve upon these designs, researchers need a platform upon which to prototype new interface ideas and test their effects. Because of the VoteBox's generalized pre-rendered interface architecture, interface designers can alter the visual appearance of the voting system without touching the code. For more sophisticated modifications, including non-visual interface experimentation, VoteBox provides a well-documented, modular Java implementation. In separate branches of the software we have added clearly-identified blocks of code for recording the precise timing and nature of the user's actions, as well as other data crucial to usability research.

4 Discussion

A full evaluation of our system is necessarily dependent upon a human factors study of efficiency and accuracy for poll workers and voters. We focus here on evaluating the scalability and security properties of a voting system built on Auditorium.

4.1 Performance

Our chief concern is security, but we must first engage in some back-of-the-envelope calculations to affirm that our system is well matched to the computational resources of a polling place. For our calculations we will assume that an individual machine casts at most one ballot every 3 minutes.

We further assume a polling place with 10 voting machines, which is the highest concentration of electronic machines per polling place in the United States found in a 2004 survey [10]. There is a trend in the United States toward "voting centers" in which many small polling places are consolidated into one large facility serving multiple precincts [39]. These centers may require hundreds of voting machines, which for scaling purposes can easily be segmented into a number of smaller disjoint Auditorium networks. Even if a polling place has only a single voting machine, perhaps due to a small precinct or perhaps because the DRE is present only for voters unable to use paper ballots, Auditorium still provides entanglement between events from the controller and the DRE. Likewise, a precinct-based optical ballot scanner could participate in the Auditorium protocol alongside a controller and a DRE.

4.1.1 Network load

The most burdensome part of our design is the all-to-all connectivity graph. Each message a node wishes to broadcast in the Auditorium results in roughly n^2 messages on the network. More exactly, the originating node sends the message to $n-1$ neighbors, each of which will forward the same message to their other neighbors, resulting in $n-2$ more messages for each. Each of these messages should be old news to its recipient by this point, and so the flood stops here. The total number of messages sent is therefore $n + (n-1)(n-2)$; we round up to n^2 (100) for our hypothetical polling place.

Assuming ballots are cast every 3 minutes on each of 10 machines, we have 200 ballots per hour, or 600 messages per hour (since each cast ballot is the result of an authorized-to-cast, cast-ballot, received-ballot message exchange). Since each Auditorium broadcast results in roughly 100 actual message transmissions, we now have 60,000 messages per hour, or about 17 messages per second. Add to this the periodic heartbeat messages, which (if issued every 5 minutes by each node) bring our total to 20 messages/sec.

Most messages should be on the order of 1 KB, so this corresponds to a maximum cross-sectional network bandwidth requirement of roughly 164 kbps, which a 10-baseT Ethernet hub can handle with plenty of headroom for other incidental messages (node join, polls closing, etc.) in the Auditorium.

Not every message fits in one or two packets, however; the authorized-to-cast message, in particular, contains a complete ballot definition, which in our prototypes (including all pre-rendered UI elements) is roughly a megabyte in size. This traffic now dominates our bandwidth calculations; 200 ballots per hour result in up to 20,000 cast-ballot messages per hour, or about 6 per second, which is about 50 megabits and requires a faster network.

If further experimentation shows that bandwidth is a problem, we offer the following optimization. Because the largest part of a ballot definition is its collection of pre-rendered images, these can be distributed to machines ahead of time. An authorized-to-cast message would then include only the logical ballot definition itself, which in turn references these images. We can certify image files by including the cryptographic hash of each image in the ballot definition (or in the image's filename itself, making images self-certifying). In this way we can be sure that even though a ballot definition does not carry its own images, the correct image will be displayed.

4.1.2 Processor load

The VoteBox is designed to run atop general-purpose computer hardware, so the computational effort needed to participate in Auditorium must be within the bounds of such a system.

We used OpenSSL's built-in `speed` microbenchmark tool on an unloaded 300 MHz Pentium II with 256 MB RAM (OpenSSL 0.9.7e compiled with `-march=i686`, Linux kernel 2.6.8). This is not a fast computer by modern standards; we use it as a model of the kind of horsepower we believe can be inexpensively provided using low-end, commodity CPUs.

Our test machine can perform 796 1024-bit RSA verifications and 42 signatures per second; it encrypts 5 MB of data per second with AES-256. SHA-1 can digest 20 MB of data per second. Based on the demands of our deployment scenario, our target hardware is more than sufficient; a deployment of Auditorium requiring substantially greater cryptographic performance could be achieved straightforwardly by provisioning hardware adequate to the task.

4.1.3 Storage

Storage is the least expensive and most plentiful resource in a VoteBox; assuming our worst-case hypothesis from Section 4.1.1, a machine might receive 5 messages per second, of which a third contain large ballot files, but because most of these are duplicates (thanks to flooding), we need store very few. We rely on our original estimate of 200 ballots per hour, or 2400 per day (assuming the polling place is open for 12 hours), which means 4800 small messages (cast-ballot and received-ballot; heartbeats are on this order of magnitude as well) and another 2400 large messages (authorized-to-cast). The large messages dwarf the smaller ones, so we have roughly 2400 MB of data to store, supportable even with solid-state flash memory. With a hard drive we have the luxury of preserving this data forever; a VoteBox with a hard drive needs no "clear" function, making it still harder to accidentally destroy election records. A caching strategy for ballot definitions, as described in Section 4.1.1, would reduce even this requirement considerably; storing only small messages now, we need on the order of a few megabytes, allowing even flash memory to operate without needing a "clear" function.

4.2 Robustness to failure and attack

The Auditorium design was prompted by irregularities and ambiguities that we discovered in a real election, as described in Section 2. We now consider the possible causes, both benign and malicious, of those problems and show how the Auditorium allows us to detect, record, and recover from them.

A1 Early machine exit. *Scenario:* A VoteBox suddenly departs the Auditorium network. The machine may be inoperable and any storage lost. *Response:* All ballots cast from the failed machine are replicated on other VoteBoxes and on the controller; they can be counted as part of the final tally. Note also that because VoteBoxes are interchangeable, a machine may be brought in from storage or from another precinct to replace the failed machine.

A2 Late machine entry. *Scenario:* A VoteBox enters the Auditorium after the election has started; it claims to have recorded no votes. (See Figure 1 for a real-life example.) *Possible causes:* (1) A machine was brought on-line during the day to assist additional voters or to replace a failed machine. (2) A machine was erased, possibly by accident, destroying legitimate votes cast earlier in the day. *Response:* We can look at the logs of other VoteBoxes and of the controller to see if the machine in question cast any votes earlier in the day. Any such votes can be safely counted in the final tally, and the cause of the machine's erasure investigated after the election. Any new votes cast by the machine after joining will be recorded as normal.

A3 Machine re-entry. *Scenario:* A VoteBox leaves the network and re-enters it some time later. *Possible causes:* Machine crash; temporary isolated power or network interruption. *Response:* The response is identical to that for early machine exit and late machine entry. The logs held by the controller and other VoteBoxes regarding the previous activity of the re-entered machine are

tamper-evident. The log on the machine in question may have missed messages during its downtime and thus may have a gap, but it will quickly re-join the global entanglement and continue to participate in voting.

A4 Extraneous cast ballots. *Scenario:* A VoteBox appears to have votes cast on the wrong day. *Possible causes:* (1) Clock set wrong. (2) Test votes accidentally considered as possible real votes. (3) Intentional, malicious attempt to subvert the correct tally by stuffing the ballot box with illegitimate votes before or after the election. *Response:* For (1), if the votes are valid, their local (erroneous) timestamps are irrelevant; they will be provable successors to the polls-open event and predecessors of the polls-closed event. The situations (2) and (3), based on our definition of legitimate votes, will be detected when the logs are analyzed. A vote cast outside of the temporal bounds of the election (e.g. a test vote or a stuffed ballot) will have no provable link to the entangled timeline, nor will it succeed a valid authorized-to-cast message, so it will be considered illegal.

A5 Electrical failure. *Scenario:* Electricity fails at the polling place. *Response:* This is a known problem with electronic voting machines of any sort. Most modern DREs have battery backups; there is no reason the controller and network switch can not also be backed up in this way (perhaps as simply as plugging them into a UPS). Battery status is part of the `heartbeat` message, so the controller can display the status of each VoteBox.

4.3 Mega attacks

We now investigate a class of possible (if implausible) threats to election integrity we term "mega attacks." These require either widespread collusion or overwhelming force in order to execute, but the risk—attackers able to exert total control over the outcome of an election—is just as extreme. Most voting systems, electronic or otherwise, are vulnerable to such attacks; our goal for voting in the Auditorium is to be able to recover from these attacks where possible, and to detect them in any case.

M1 Switched results. *Scenario:* The night before the election, parties in control of all election hardware use the hardware to conduct a secret election with a chosen outcome. On election day, voters cast ballots as normal, but the attackers substitute the results of the secret election (including cast ballots and entangled logs) when the polls are closed. *Response:* This attack is equally effective against paper ballots, and should be addressed with human procedures. For example, no single party should be allowed unsupervised custody of the machines to be used on election day. Alternatively, distribution of controllers should be delayed until election morning.

We can also use the properties of the Auditorium to detect this attack. On the morning of the election, the election administrator can distribute to each polling place a nonce to be input into each controller. This nonce, hard to guess but easy to input, might take the form of a few English words. The controller would require the user to input the nonce before opening the polls; the nonce would then be embedded in the polls-open message. Any audit can examine the polls-open message to see if the nonce is correct (modulo minor keying errors by the poll workers). A sleeopver conspiracy would be unable to guess the correct nonce to inject into their polls-open message, and thus the Auditorium record of their secret election will be detectably invalid.

A similar defense will thwart attackers who run a secret election and switch the results *after* election day. We do this by immediately publishing the hash value of the polls-closed message. Copies may go to election observers, newspapers, and so forth. This effectively seals the results of the legitimate election, making it impossible to substitute new results later.

M2 Shadow election. *Scenario:* Similar to the switched-results attack, a conspiracy of election workers substitutes false election results for the real ones. Instead of conducting the election the night before, they conduct the secret election on election day as a "shadow" of the real one, so they have access to any nonces used to validate the date of the election. *Response:* In order to conduct a shadow election, the attackers will need to duplicate the entire voting apparatus, down to the private keys used by each VoteBox to sign messages. (Creating new keypairs won't work; the correct keys are enclosed in certificates signed by election officials.) To make this duplication difficult, we must investigate hardware-based protection schemes, such as trusted platform module (TPM) chips which resist extraction of key material. We discuss TPM further in Section 5.2.

M3 Booth capture. *Scenario:* Armed attackers take control of the polling place by force and stuff ballots until the police arrive. *Response:* Western readers may find this attack implausible, but such events are not uncommon in fledgling democracies and have occurred in India as recently as 2004 [31]. Attackers have two potential goals: (1) cast fraudulent ballots; (2) destroy legitimate ballots. We can mitigate the danger of (1) by estimating when the attack took place (perhaps we allow poll workers to broadcast an "election compromised" Auditorium message, rather like an alarm button at a bank) and discarding votes cast after that point.

In the case of (2), we can recover votes from any machine not destroyed, but we cannot recover from complete destruction of the polling place. The only defense would be a network link to an offsite location (over which Auditorium messages would be broadcast, just as within

the polling place). By removing the air-gap between the precinct network and the "outside world," we greatly increase the attack surface of the polling place, thereby introducing unacceptable risk.

In either case, this attack is trivially detectable, if not always recoverable.

4.4 Software tampering

Finally we consider software tampering, a critical issue with any form of electronic voting. Though this work focuses on issues of auditability arising when correct voting systems fail or are used incorrectly, we must consider how to deal with the introduction of malicious code into the overall system.

Our design for Auditorium makes it very hard for a malicious node to corrupt the entangled record. Hash chaining prevents insertion of spurious events; digital signatures prevent forgery. Digital signatures also allow VoteBoxes to unambiguously identify the source of each message, so a node wishing to deny service by, for example, filling up the disks of its peers with junk messages can be effectively ignored by other nodes. As shown in Section 3.5, the gossip protocol of the Auditorium makes it impossible for a malicious node to maintain multiple divergent timelines without being detected. Any of the above events or other unusual activity, if found in the audit logs, will necessitate impounding and further investigation of the equipment used in the election.

Malicious software on a single VoteBox would not be able to cast unattended votes without a corrupt controller, as only the controller can generate the necessary `authorized-to-cast` message. Of course, the VoteBox could show the voter his or her correct selection while quietly casting a vote for somebody else. Addressing this concern requires mechanisms beyond the Auditorium, such as a trusted platform module (TPM, described in Section 5.2), voter-verified paper ballots (VVPB), or end-to-end cryptographic verification techniques (see Section 5.3). All of these techniques are *complementary* to Auditorium, and can easily integrate with our system.

5 Future work

5.1 Human factors research

As we have found in practice, anomalous election events can be caused by a failure to follow proper procedures. If incorrect operation is impossible but correct operation is also challenging, our system is still a failure, so we must also investigate the usability of VoteBoxes and controllers.

From the poll worker's perspective there is even more to study: we must design and evaluate the interface of the controller. We intend to enumerate the tasks a poll worker must perform, and then test human subjects for efficiency and accuracy at completing those tasks. We anticipate Auditorium will improve poll worker compliance with procedures due to the console's global view on voting machine state and ability to assist in polling place supervision.

5.2 Trusted computing

The use of trusted-computing technologies like the Trusted Platform Module (TPM) offers us an opportunity to reason at runtime about the configuration of the machine. We can use it to securely boot the system into a known, trusted configuration [2] or to deny crucial services and data (such as cryptographic keys) to a corrupt or uncertified system [25, 9]. By requiring a signed, fully-certified software stack in order to boot a VoteBox, election officials can prevent uncertified software from being used on election day (a problem that arose in California in 2003 [46]).

A TPM also allows us to protect sensitive data from tampering, even by our VoteBox software. We can use the TPM as a secure co-processor, signing Auditorium entries without divulging key material. Secure registers in the TPM might be used to hold data such as the VoteBox machine ID, or the machine's protected count; the TPM 1.2 standard specifies a monotonic counter [40] that may suffice.

A fundamental feature of a TPM is its ability to generate *attestations* (i.e., cryptographically signed statements) as to the present state of the computer system. These attestations can be broadcast, perhaps as part of the `heartbeat` messages, allowing Auditorium hosts to log each others' known software state. We could also leverage purely software attestation systems, such as Pioneer [37, 36], where machines will challenge one another to perform computations based on their state and will carefully time the results. If a VoteBox fails to properly attest to its state, then its votes may well be corrupt. While election officials would need to decide what to do with those records, the controller could detect this situation on election day and subsequently refuse to authorize any new votes on the suspect machine.

5.3 Cryptography

While our system makes use of well-understood cryptographic operations such as one-way hash functions and public-key signatures and encryption, we believe that more exotic cryptography may be added to our system to provide additional properties. For example, threshold encryption [38] can be employed to further protect sealed cast ballots. In such a scenario, at least one share of the decryption key should be held by a party outside the polling place, to limit a local attacker's ability to compromise the anonymity of the vote.

The Auditorium is also a strong complement to other

cryptographic voting techniques. It is useful in any design that demands a ballot broadcast channel or "bulletin board." Additionally, it is compatible with voting systems that improve voter verifiability, including voter-visible cryptographic ballot systems such as those proposed by Chaum [13, 14] and Rivest [30]. Our system could also leverage techniques, such as Adida and Wikström [1], using homomorphic encryption [15, 7, 3] and verifiable mixes [12, 32, 27] to securely derive sorted or shuffled plaintext ballots from our ordered ciphertext ballots. This mixing process could be performed after the election was over, by "trustees" of the election, or it could be performed by the VoteBoxes in the polling place, depending on when and where it was deemed necessary to have access to plaintext ballots.

A related but more fundamental problem is the use of non-determinism in cryptography. If the same plaintext always yields the same ciphertext, that would damage the ability of the cryptography to hide a voter's intent. On the other hand, if a random number is used to initialize the crypto system, as is standard practice, this random number could then serve as a *subliminal channel*, allowing a malicious machine to reveal something about the voter's intent. A suitable solution is to embed the cryptosystem in trusted hardware, where software tampering cannot affect it. Many TPM chips offer the appropriate functionality.

5.4 Auditorium applications

Voting is the first application of the Auditorium infrastructure. We intend to investigate other domains in which small groups of nodes need to share a global, trustworthy perspective on fine-grained system state changes or other ordered events. For example, collaboration software could take advantage of Auditorium to mediate simultaneous changes by different users. We have prototyped a collaborative word processor called Editorium that uses a timeline of editing operations to allow authors to work together on a single document.

More generally, any application which can leverage a secure shared channel for distributing temporally-sensitive information can potentially leverage Auditorium. For example, the CATS accountable network storage system [45] requires such a publishing medium for disseminating state commitments of participants; the authors suggest a possible implementation based on gossip and secure broadcast, which is a perfect match for Auditorium.

6 Conclusion

We stress that our work is a complement to other ongoing voting security research. With the Auditorium we do not attempt to solve the problem of untrusted software, although trusted computing technologies, voter-verifiable paper ballots, and good software design practices can all be combined with Auditorium to address it. Nor do we address the problem of justifying to voters that their ballots are counted as cast.

Instead, we have focused on a problem that is not merely theoretical: that of auditing the procedures followed and events encountered in a polling place on election day. *Secure logs* allow us to reason about the time at which events occurred. *Entanglement* allows us to extend this reasoning to the entire polling place, and help us to fix the events of any one voting machine in the timeframe of another (for example, the election controller). *Broadcast* and *replication* offer us the opportunity to capture auditing data in many places, providing many redundant accounts of the course of the election. With these techniques, we can be sure that auditing data heard in the Auditorium is able to survive a number of quite common polling place failures (as well as a number of uncommon ones). Perhaps most importantly, our design can be added to electronic voting systems in use today, including those used in the election whose auditing anomalies inspired our work.

Acknowledgments

We would like to thank Mike Byrne, Sarah Everett, and Kristen Greene for their design of the prototype VoteBox interface and for their continued feedback as they use the software for their research. Thanks also to Kyle Derr and Ted Torous for their contributions to the development of VoteBox. We thank Ben Adida for his insightful notes on our design; it is his suggestion that Auditorium may be useful as a broadcast channel for cryptographic elections. We also owe thanks to the law firm of Campero & Becerra in Laredo, TX and Oscar Villarreal, Webb County Elections Administrator, for giving us the rare opportunity to examine the actual voting systems used in a contested election. Finally, we thank the anonymous reviewers of this paper for their helpful suggestions and comments.

This work is supported, in part, by generous gifts from Microsoft and Schlumberger as well as NSF CNS-0524211 (the ACCURATE research center).

References

[1] Adida, B., and Wikström, D. How to shuffle in public. In *Theory of Cryptography, Proceedings of TCC 2007* (February 2007), Lecture Notes in Computer Science, Springer-Verlag, pp. 555–574.

[2] Arbaugh, W. A., Farber, D. J., and Smith, J. M. A secure and reliable bootstrap architecture. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy* (Oakland, CA, May 1997).

[3] Baudron, O., Fouque, P.-A., Pointcheval, D., Stern, J., and Poupard, G. Practical multi-candidate election system. In *PODC '01: Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing* (Newport, RI, 2001), pp. 274–283.

[4] Bederson, B. B., Lee, B., Sherman, R. M., Herrnson, P. S., and Niemi, R. G. Electronic voting system usability issues. In *CHI '03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Ft. Lauderdale, FL, 2003), pp. 145–152.

[5] Bellare, M., and Yee, B. Forward integrity for secure audit logs. Tech. rep., UC at San Diego, Dept. of Computer Science and Engineering, Nov. 1997.

[6] Benaloh, J. Simple verifiable elections. In *USENIX/ACCURATE Electronic Voting Technology Workshop* (Vancouver, B.C., Canada, June 2006).

[7] Benaloh, J., and Yung, M. Distributing the power of a government to enhance the privacy of voters. In *5th ACM Symposium on Principles of Distributed Computing (PODC)* (Calgary, Alberta, Canada, 1986), pp. 52–62.

[8] Bethencourt, J., Boneh, D., and Waters, B. Cryptographic methods for storing ballots on a voting machine. In *14th Network and Distributed System Security Symposium (NDSS)* (San Diego, CA, Feb. 2007).

[9] BitLocker Drive Encryption: Technical Overview, Apr. 2006. http://www.microsoft.com/technet/windowsvista/security/bittech.mspx.

[10] Brace, K. W., and McDonald, M. P. Final Report of the 2004 Election Day Survey. Submitted to the U.S. Election Assistance Commission, Sept. 2005. http://www.eac.gov/election_survey_2004/toc.htm.

[11] Brennan Center Task Force on Voting System Security. *The Machinery of Democracy: Protecting Elections in an Electronic World*, June 2006. http://www.brennancenter.org/dynamic/subpages/download_file_36343.pdf.

[12] Chaum, D. Untraceable electronic mail, return addresses and digital pseudo-nyms. *Communications of the ACM 24*, 2 (1981), 84–88.

[13] Chaum, D. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy 2*, 1 (Jan. 2004), 38–47.

[14] Chaum, D., Ryan, P. Y. A., and Schneider, S. A. A practical, voter-verifiable election scheme. In *ESORICS '05* (Milan, Italy, 2005), pp. 118–139.

[15] Cohen (Benaloh), J., and Fischer, M. A robust and verifiable cryptographically secure election scheme. In *28th IEEE Symposium on Foundations of Computer Science (FOCS)* (Portland, OR, 1985), pp. 372–382.

[16] Everett, S. P. *The Usability of Electronic Voting Machines and How Votes Can Be Changed Without Detection*. PhD thesis, Rice University, Houston, TX, 2007.

[17] Everett, S. P., Byrne, M. D., and Greene, K. K. Measuring the usability of paper ballots: Efficiency, effectiveness, and satisfaction. In *Proceedings of the Human Factors and Ergonomics Society 50th Annual Meeting* (Santa Monica, CA, 2006).

[18] Feldman, A. J., Halderman, J. A., and Felten, E. W. Security analysis of the Diebold AccuVote-TS voting machine. In *Proceedings of the 2nd USENIX/ACCURATE Electronic Voting Technology Workshop* (Boston, MA, Aug. 2007).

[19] Gonggrijp, R., and Hengeveld, W.-J. Studying the Nedap/Groenendaal ES3B Voting Computer: A computer security perspective. In *Proceedings of the 2nd USENIX/ACCURATE Electronic Voting Technology Workshop* (Boston, MA, Aug. 2007).

[20] Greene, K. K., Byrne, M. D., and Everett, S. P. A comparison of usability between voting methods. In *USENIX/ACCURATE Electronic Voting Technology Workshop* (Vancouver, B.C., Canada, 2006).

[21] Hursti, H. Critical security issues with Diebold TSx. Tech. rep., Black Box Voting, May 2006. http://www.blackboxvoting.org/BBVtsxstudy.pdf.

[22] Kohno, T., Stubblefield, A., Rubin, A. D., and Wallach, D. S. Analysis of an electronic voting system. In *Proc. of IEEE Symposium on Security & Privacy* (Oakland, CA, 2004).

[23] Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21*, 7 (1978), 558–565.

[24] Maniatis, P., and Baker, M. Secure history preservation through timeline entanglement. In *Proceedings of the 11th USENIX Security Symposium* (San Francisco, CA, Aug. 2002).

[25] Marchesini, J., Smith, S., Wild, O., and MacDonald, R. Experimenting with TCPA/TCG Hardware, Or: How I Learned to Stop Worrying and Love The Bear. Tech. Rep. TR2003-476, Department of Computer Science, Dartmouth College, Dec. 2003.

[26] Molnar, D., Kohno, T., Sastry, N., and Wagner, D. 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* (Oakland, CA, May 2006).

[27] Neff, C. A. A verifiable secret shuffle and its application to e-voting. In *CCS '01: Proceedings of the 8th ACM Conference on Computer and Communications Security* (Philadelphia, PA, 2001), pp. 116–125.

[28] Pope John Paul II. *Universi Dominici Gregis*: On the vacancy of the Apostolic See and the election of the Roman Pontiff. Chapter V: The Election Procedure, Feb. 1996.

[29] RABA Technologies. *Trusted Agent Report: Diebold AccuVote-TS Voting System*. Columbia, MD, Jan. 2004. http://www.raba.com/press/TA_Report_AccuVote.pdf.

[30] Rivest, R. L. The ThreeBallot voting system. http://theory.csail.mit.edu/~rivest/Rivest-TheThreeBallotVotingSystem.pdf, Oct. 2006.

[31] Rohde, D. On New Voting Machine, the Same Old Fraud. *The New York Times* (April 27, 2004). http://www.nytimes.com/2004/04/27/international/asia/27indi.html.

[32] Sako, K., and Kilian, J. Secure voting using partially compatible homomorphisms. In *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 1994), Springer-Verlag, pp. 411–424.

[33] Sastry, N., Kohno, T., and Wagner, D. Designing voting machines for verification. In *Proceedings of the 15th USENIX Security Symposium* (Vancouver, B.C., Canada, Aug. 2006).

[34] Schneier, B. Hacking the papal election, April 14, 2005. http://schneier.com/blog/archives/2005/04/hacking_the_pap.html.

[35] Schneier, B., and Kelsey, J. Cryptographic support for secure logs on untrusted machines. In *USENIX Security Symposium* (San Antonio, TX, Jan. 1998), pp. 53–62.

[36] Seshadri, A., Luk, M., Perrig, A., van Doorn, L., and Khosla, P. Externally verifiable code execution. *Communications of the ACM 49*, 9 (Sept. 2006), 45–49.

[37] Seshadri, A., Luk, M., Shi, E., Perrig, A., van Doorn, L., and Khosla, P. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. In *ACM Symposium on Operating Systems Principles (SOSP 2005)* (Brighton, U.K., Dec. 2005).

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

[39] Stein, R. M., and Vonnahme, G. Election day vote centers and voter turnout. Prepared for presentation at the 2006 Annual Meetings of the Midwest Political Science Association, Apr. 2006.

[40] Trusted Computing Group. TPM v1.2 Specification Changes, Oct. 2003. https://www.trustedcomputinggroup.org/downloads/TPM_1_2_Changes_final.pdf.

[41] Wagner, D., Jefferson, D., Bishop, M., Karlof, C., and Sastry, N. Security analysis of the Diebold AccuBasic interpreter. California Secretary of State's Voting Systems Technology Assessment Advisory Board (VSTAAB), Feb. 2006. http://www.ss.ca.gov/elections/voting_systems/security_analysis_of_the_diebold_accubasic_interpreter.pdf.

[42] Wallach, D. S. Security and Reliability of Webb County's ES&S Voting System and the March '06 Primary Election. Expert Report in *Flores v. Lopez*, http://accurate-voting.org/wp-content/uploads/2006/09/webb-report2.pdf, May 2006.

[43] Yasinsac, A., Wagner, D., Bishop, M., Baker, T., de Medeiros, B., Tyson, G., Shamos, M., and Burmester, M. *Software Review and Security Analysis of the ES&S iVotronic 8.0.1.2 Voting Machine Firmware*. Florida State University, Tallahassee, FL, Feb. 2007. http://election.dos.state.fl.us/pdf/FinalAudRepSAIT.pdf.

[44] Yee, K.-P., Wagner, D., Hearst, M., and Bellovin, S. M. Prerendered user interfaces for higher-assurance electronic voting. In *USENIX/ACCURATE Electronic Voting Technology Workshop* (Vancouver, B.C., Canada, 2006).

[45] Yumerefendi, A. R., and Chase, J. S. Strong accountability for network storage. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST '07)* (San Jose, CA, Feb. 2007), pp. 77—92.

[46] Zetter, K. E-voting undermined by sloppiness. *Wired News* (Dec. 17, 2003). http://wired.com/news/politics/evote/0,61637-0.html.