TPM Meets DRE: Reducing the Trust Base for Electronic Voting Using Trusted Platform Modules

Russell A. Fink, Alan T. Sherman, and Richard Carback

***Abstract*—We reduce the required trusted computing base for direct recording electronic (DRE) voting machines with a design based on trusted platform modules (TPMs). Our approach ensures election data integrity by binding the voter's choices with the presented ballot using a platform vote ballot (PVB) signature key managed by the TPM. The TPM can use the PVB key only when static measurements of the software reflect an uncompromised state and when a precinct judge enters a special password revealed on election day. Using the PVB with the TPM can expose authorized software, ballot modifications, vote tampering, and creation of fake election records early in the election process. Our protocol places trust in tamper resistant hardware, not in mutable system software. Although we are not the first to suggest using TPMs in voting, we are the first to provide a detailed engineering protocol that binds the voter choices with the presented ballot and uses the TPM to enforce election policy. We present the protocol, architecture, assumptions, and security arguments in enough detail to support further analysis or implementation.**

***Index Terms*—Applications, direct recording electronic (DRE), electronic voting, protocols, security and privacy analysis, system design and implementation, trusted platform modules (TPMs).**

I. Introduction

Direct recording electronic (DRE) voting machines can offer many compelling benefits, including good usability and accessibility, support of multiple ballots and languages, and elimination of overvotes and unintentional undervotes [14]. Unfortunately, bad security engineering of existing products (e.g., [2]–[4], [10]) has largely discredited the entire approach along with many of its strengths. We offer an approach to DRE design based on trusted cryptographic hardware that offers a much more secure way to build DREs while preserving their advantages. This engineering paper describes in considerable detail how to design a more trustworthy DRE for protecting vote data integrity and ballot privacy. Our protocol is a first step in our larger effort to apply high-assurance computing techniques to voting technology.

We have designed a protocol for election systems that secures data with private signature keys managed by special physically secured hardware resident on commercial PC computing platforms. Our protocol uses the *trusted platform module (TPM)*—an embedded processor that provides cryptographic services, stores measurements of booted software, and manages onboard nonvolatile memory—to create and manage a special signature key called the *platform vote ballot (PVB)* key. The PVB binds together the booted state of the platform, the ballot presented to the voter, and the voter's cast vote, thwarting unauthorized vote modification, insertion, or deletion.

In our protocol, the PVB key is created and bound to the correct platform state during the initial DRE software load, and is unlocked by a password revealed on election day. During the polling phase, the TPM signs a hash of each recorded vote and ballot with the PVB private key. Votes are recorded in pseudorandomly determined storage slots, and the storage is signed by the PVB after each recorded vote. At the close of the polls, tallying officials receive the signed storage and verify the signatures of both the individual votes and the storage area using the PVB public key. Verification ensures that the DREs booted the correct software, voters used the correct ballots, and the votes were not modified, omitted, or illegally inserted or deleted. In this paper, we present our protocol with enough detail to demonstrate feasibility of an actual implementation on a system compliant with the *Trusted Computing Group (TCG) software stack (TSS)* [36].

Although others have suggested using TPMs for voting, our protocol is the first to use a TPM to bind the ballot, vote data, and storage integrity to the platform state to achieve these goals:

1) Hardware-based protection of keys—the plaintext PVB signature key is never revealed outside of the TPM, preventing errant or malicious disclosure of the private key.
2) Cryptographic binding of vote to ballot—verification that a specific ballot guided the voter's decisions.
3) Hardware-based software state and election policy enforcement—the TPM requires proper platform software measurements and election initiation passwords to store valid data (resisting day-before attacks[1]).
4) Cryptographic integrity and privacy—integrity is enforced by public key cryptography, and privacy is preserved by a pseudorandom ordering of votes in storage.

While the TPM and our protocol improve system-level assurance in electronic voting, we acknowledge valid criticisms about current DRE implementations and many electronic voting systems. Electronic voting requires voters to have faith in the correct operations of the system hardware and software: DRE

Manuscript received February 15, 2009; revised September 30, 2009. First published October 20, 2009; current version published November 18, 2009. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Bart Preneel.

R. A. Fink is with the Johns Hopkins University/Applied Physics Laboratory, Laurel, MD 20723 USA, and also with the Cyber Defense Lab, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, MD 21250 USA (e-mail Russ.Fink@jhuapl.edu).

A. T. Sherman is with the Cyber Defense Lab, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, MD 21250 USA, and also with the National Center for the Study of Elections, University of Maryland, Baltimore County, Baltimore, MD 21250 USA (e-mail: sherman@umbc.edu).

R. Carback is with the Cyber Defense Lab, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, MD 21250 USA (e-mail: carback1@umbc.edu).

Digital Object Identifier 10.1109/TIFS.2009.2034900

[1] On June 28, 2009, the day Honduras President Manuel Zelaya was ousted, officials found certified election results on government computers for an election that was to have taken place that day [9].

1556-6013/$26.00 © 2009 IEEE

users implicitly trust the CPU, system RAM, and user interface peripherals (touch screen or other input device), and no random testing paradigm that we know of includes hardware component analysis. Electronic data capture systems—including electronic voting systems—mask their inner workings, in that a user interacting with a computer cannot see the computation taking place or know what bits were recorded on the media. While our protocol enables the tallying authority to detect integrity problems with the software and data, the present design does not allow the voter to interactively verify proper capture, processing, or storage integrity of her vote while the polls are open. Despite avoiding the significant expense of printing paper ballots, electronic voting systems can mean significant up-front costs for procurement, installation, training, upgrade and maintenance, reflecting a high cost per user. Improved security engineering and implementation, a reduced trust base, and better transparency and verifiability are required to alleviate concerns of traditional DRE and electronic voting.

Apart from the risks, DRE systems provide good usability features. DRE systems readily support multilanguage ballots, multiple ballots for different races, lengthy ballots with many races or candidates, and a variety of input methods and display modes, including general use touch screens and also audible and sip-and-puff options for disabled voters. They can support the option to use innovative presentation techniques such as randomly ordering the candidate lists to avoid the candidate *primacy* phenomenon where a candidate receives an unusually high number of votes based on appearing as the first candidate in a list [20]. Although not always appropriate or currently permitted by law, there are many circumstances in which such innovative techniques can lead to more accurate capture of voter will. Clarity of intent is most accurately captured by digital means, avoiding ambiguous user markings such as dimpled chads, markings from butterfly layouts, or incorrect marks on paper-based forms.

DRE terminals can protect the vote records with digital signatures prior to being offloaded for tallying, reducing risk in the chain of custody, whereas paper ballots are subject to omission, loss, or tampering. While our protocol makes certain assumptions about the hardware (stated in Section IV-B), it uses trusted hardware to overcome many of the software risks of current DRE systems, thus reducing the overall size of the trusted computing base. Our vision is that the hardware cryptographic primitives of TPMs can help improve DRE systems with commercial computing components for a reasonable cost, allowing DRE benefits with fewer security risks.

This paper is organized as follows: Section II briefly reviews previous and related work; Section III introduces the capabilities of a TPM; Section IV describes a notional system architecture and states several security assumptions; Section V presents our protocol; Section VI analyzes the protocol, including its security and special features; Section VII discusses benefits and limitations; Section VIII presents future work; and Section IX concludes our current work. We assume that the reader is familiar with some high-level principles of cryptography including digital signatures, hashing, and encryption, and also a general voter's knowledge of elections and election procedures. The paper is of interest both for applying trusted hardware to voting and as an example of building secure systems with trust rooted in TPMs.

II. Previous and Related Work

Arbaugh [1] suggested using TPMs in voting by outlining an on-line protocol for attesting systems through a central server. Rössler, *et al.* [25] proposed using hardware security modules in postal-voting where each voter submits a ballot encrypted with a public key to the tallying server. Both approaches seem promising, but omit key design details. Paul and Tanenbaum [22] sketched a voting system architecture incorporating TPMs, but the TPMs' role assures only presence of correct software—the platform state is not bound to the cast ballot.

Yee [37] designed a DRE with a greatly reduced trusted code base to simplify software inspections, but inspections cannot prevent malicious tampering of the DRE immediately prior to operations.

The Scytl architecture created by Jorba, *et al.*, described with few details in [16], suggests using a hardware security module to protect chained digital signatures but not signature keys, and uses light-weight voting software booted from a CD-ROM to eliminate reliance on preinstalled software and hardware. The security of any system that obtains software and private keys from removable media is vulnerable to compromise through theft and replacement of the media. Our approach stores and uses private keys only in tamper-resistant hardware, preventing theft or unauthorized disclosure of the keys.

Feldman, *et al.* [10] suggested using technology from the TCG, cautioning that this technology "could not prevent malicious code from changing future votes by altering data before it is sent to the storage device." Our approach uses a hardware root of trust making it harder to inject malicious software, but we rely on software correctly taking measurements and correctly executing the voting features. Furthermore, because the TPM signs each cast ballot, malicious software cannot modify a vote (without detection) once it has been processed by the TPM.

TPMs are described in the specifications [36]. Pearson *et al.* give a slightly dated but comprehensive overview of TPMs and the TCG [23], and Challener [5] provides an excellent practical guide to the TPM for software developers. Additionally, TrouSerS [35] is an open source implementation of the TSS and includes test suite software useful for understanding the programming interface, while Strasser [34] provides an open source TPM emulator to aid development.

Previous authors have applied TPMs to nonvoting domains. Sevinc [31] described a key distribution protocol that sends secrets from a server to a TPM-enabled client, but the server has no way to attest the software state of the client. Our protocol binds the PVB key to the software state of the DRE allowing the election authority to verify the correct configuration of the DRE.

Previous studies on security and implementation problems of current DRE systems include Kelsey's [17] catalog of DRE attack strategies, a threat analysis derived from attack trees by the Brennan Center [21], and research analyses by Kohno *et al.* [18], SAIC [28], RABA [24], and Compuware Corporation [8] on flawed commercial DRE implementations. Additionally, Hursti [15] analyzed the problems of unauthenticated software installs, and Feldman, *et al.* [10] analyzed the damage caused by viruses when policies and procedures are not followed. Additional vulnerabilities in modern DREs were uncovered in the EVEREST Project [4] and in the California Top-To-Bottom Review [3].

Unfortunately, most critics of current DRE systems do not offer a high-integrity alternative. Some groups advocate using so-called "voter-verified" systems, such as precinct-count optical scan or *voter verifiable paper audit trails (VVPATs)* (e.g., [12]). However, such systems provide weak ballot custody assurance and hence offer no guarantee that the ballot verified by the voter was the ballot actually tallied. Furthermore, such systems offer poor verification guarantees for visually disabled voters.

End-to-end voter verifiable systems (E2E) (e.g., [6], [7]) provide strong assurance to the voter that her vote was cast as intended, and counted as cast, and allow independent universal verification of the election result. Our present design does not give integrity assurance to the voter in the polling location, but it does offer security assurance to the election authority that the correct software was installed, that voters used the correct ballot, and that votes were securely stored and transmitted to the central tallying location. Further, our design can detect malicious installed software in the polling booth, catching persistent software injection attacks early. Our approach could complement E2E systems by adding prompt detection of unauthorized platform software—safeguarding voter privacy—and leading to a hybrid system with a more secure electronic interface coupled with E2E voter verifiable results [11].

Some researchers have raised questions of how trustworthy TPMs are. The TPM specifications can be difficult to understand, and as a result, implementation problems can occur. Sadeghi *et al.* performed a detailed compliance analysis of five TPM implementations; of the three found to be noncompliant, only one of those implemented the current version (1.2) of the TPM specification [27]. Although correct TPM implementation is critical to a protocol like ours, there are many different TPM vendors from which to choose, and the specific problems that Sadeghi identified have no impact on our protocol.

A DRE system could be built on today's commercial platforms without hardware controlled key management or software attestation, but such systems would rely on software and procedures to prevent key theft, data modification, malicious software injection, and privacy loss. Keys can be stolen quite easily through physical attacks [13]; data can be modified on disk without added protections such as full disk encryption [30]; and rogue software can be loaded in a way that is undetectable by antivirus products [26] compromising privacy and data integrity. Trusted hardware can enhance many voting technologies by providing a secure place to store keys and the ability to attest the software state of the platform cryptographically.

III. TPM Capabilities

List of Acronyms

ITA	Independent testing authority.
NV	Nonvolatile (storage).
PCRs	Platform configuration registers.
PVB	Platform vote ballot binding key.
SRK	Storage root key.
TEA	Trusted election authority.
TPJ	Trusted precinct judge.
TPM	Trusted platform module.
TTA	Trusted tallying authority.

The TPM is an embedded cryptographic processor and nonvolatile storage device meeting specifications provided by the Trusted Computing Group [36]. The TPM allows platform attestation whereby sensitive information and keys can be bound to measured states of booted software. Version 1.2 of the TPM implements asymmetric RSA 2048-bit encryption, random number and key generation, and the Secure Hash Algorithm (SHA-1) within tamper-resistant hardware.[2] The TPM offers three major protections for computing platforms: physical and architectural security, a flexible key hierarchy, and attestation support.

The physical security of the TPM is bolstered by tamper-evident packaging and secure mounting on the motherboard. TPM-enabled architectures are designed to use the TPM as a slave device—some attach the TPM to the *low pincount (LPC)* bus, for example—ensuring that the TPM cannot interfere with or control the CPU. Sensitive data exchanged between the CPU and TPM can be encrypted to resist attacks by malicious devices or hardware probes [36].

The TPM maintains a hierarchy of keys, rooted by the *storage root key (SRK)* that protects keys and other data stored externally. The SRK is generated whenever someone takes *ownership* of the TPM, preventing new owners from accessing the former owner's keys. The SRK is generated and stored internally such that the private part never leaves the TPM. Keys in the TPM hierarchy have a *parent/child* relationship, whereby a *parent key* encrypts—or wraps—a *child key*. Children of the SRK (but not the SRK) can be exported in encrypted *blobs*, easing the storage burden on the TPM while allowing software to *load* keys back into the TPM when needed. In our protocol, the PVB is created as a child of the SRK. As additional protection, the TPM can enforce passwords on blobs to prevent unauthorized loading.

Ownership is enforced by a password that allows the *owner* to use all features of the TPM. Occasionally, an administrator other than the TPM owner needs the ability to perform limited tasks, such as invalidating a key, but the owner does not trust the administrator with the owner password. The TPM supports *delegation* to address this situation by granting limited privileges to an admininistrator enforced by a distinct password.

The TPM can support system software *attestation* by recording the state of the platform's software processes as measured through means of cryptographic hashes of executable binary code. Such measurements can be compared to the expected state to verify the platform's integrity. Measurements are done typically at the initial boot of the platform by following a sequence of load, measure, and execute steps among the major software components. For instance, the BIOS loads the boot loader, measures it, stores the measurements in the TPM, and then transfers control to it. This *measurement chain* must include all critical system components and configuration data. Measurements are stored in TPM *platform configuration registers (PCRs)*, special volatile memory locations in the TPM that either can be reset to zero, or *extended* by hashing a new value with the existing PCR value. PCRs never can be set to a specific nonzero value, for security against malicious software. A signing key can be *bound* to a particular software state by specifying acceptable values for the PCRs, allowing the TPM

[2] SHA-1's replacement will be addressed in the next version of the TPM specification.

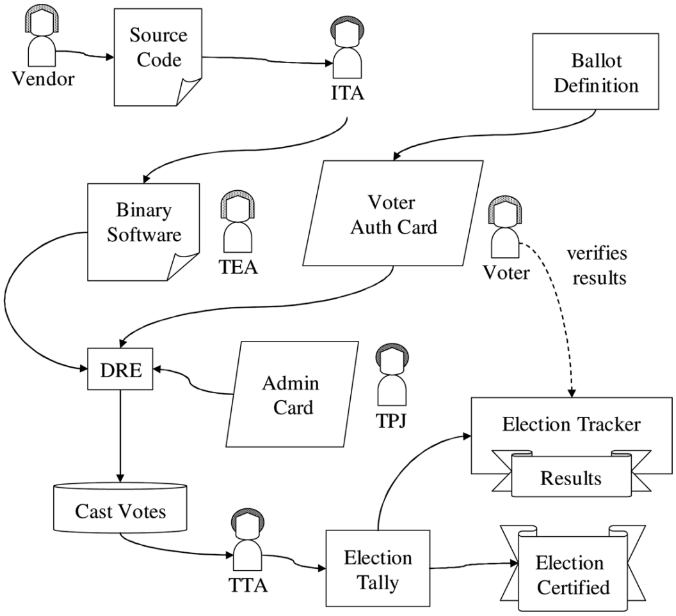

Fig. 1. Architecture includes: DRE (with a TPM), tallying systems, and tracker/reporting systems such as Election Tracker [32]; trusted authorities: Trusted Election Authority (TEA), Precinct Judge (TPJ), Tallying Authority (TTA), Independent Testing Authority (ITA); the voter; and binary images of software, ballot, and storage for cast votes.

to load the key only when the software measurements stored in the PCRs are correct.

IV. Architecture

We define a notional architecture consisting of high-level system elements, including hardware, actors, dependencies, and some security assumptions. The architecture is patterned deliberately after existing DRE architectures to show the applicability of the protocol to current technology.

A. System Elements

Fig. 1 shows the system elements in the context of the high level architecture.

B. Security Assumptions

The protocol protects the cast *vote* (recorded intent of the voter) and evidence of the *ballot* (the choices presented to the voter) provided that certain assumptions hold.

1) Asymmetric digital signature keys and hashing afforded by the TPM adequately ensure data authenticity and integrity during storage and transmittal.
2) The chain of trust of PCR measurements includes all relevant software, firmware, and configuration files, including the operating system kernel, software drivers, loadable modules, relevant dynamic libraries, the voting system software and configuration files, and relevant portions of the BIOS.
3) The TPM and other trusted hardware are operating correctly.
4) The software components that form the measured chain of trust are behaving as expected (correctly implemented, correctly executing).
5) A pseudorandom index is sufficient to protect voter identities against analysis of stored vote order.
6) A trusted hardware path exists between the DRE motherboard/CPU and all other hardware components including the screen, hard drive, external storage connections, peripherals, and input devices.
7) System memory is unmodifiable by on-board devices.

Further, the protocol requires that binaries are correctly generated from reviewed software and securely transferred between the vendor, ITA, and TEA; and that the ballots are reviewed for accuracy.

C. System Roles

The protocol trusts certain human roles to carry out parts of the election process. We take the word trust to mean an expectation of a certain behavior for a particular purpose; that is, if a trusted role behaves errantly, the security claims of the protocol no longer hold. These roles are more fully explained in the protocol steps, but a short summary follows:

1) *Trusted Election Authority (TEA)*—charged with ensuring integrity of the election and its procedures, and entrusted to protect voter privacy. Responsible for approving software and slates, initializing the election system and voting units, creating cryptographic keys and protecting the TPM key creation passwords (owner password). The TEA is critical to the entire protocol.
2) *Trusted Tallying Authority (TTA)*—receives encrypted, completed ballots, tallies them, and produces the general results.
3) *Trusted Precinct Judge (TPJ)*—primary polling location worker who activates and shuts down the DREs, enforces election rules at the polling location, and resolves problems detected by the TPM and voting software. The TPJ is trusted to help resolve problems in the precinct, taking action if the TPM refuses to load the PVB key, putting a backup machine into operation. Trust is less strict here as the protocol limits the set of cryptographic operations that the TPJ can conduct. (Denial of service is still possible by a rogue or poorly trained TPJ.)
4) *Independent Testing Authority (ITA)*—tests vendor-supplied software for compliance to specifications, performs random machine testing to ensure quality of hardware and other components outside of the protective boundary of the TPM.

V. Protocol

We present a protocol for platform and data binding of electronic data captured at the DRE during election time, and describe its assurance and security properties. The protocol will be described in several steps that tie closely with a typical election timeline, to aid understanding in how an actual implementation might be executed. The protocol provides integrity and authenticity of ballot data recorded electronically at the DRE, and utilizes the main features of the TPM. The central work of the protocol is management of the PVB key. Fig. 2 highlights the main features of the protocol.

A. Detailed Description

We now present the full details of the protocol broken into distinct voting phases. For clarity, we focus on the cryptographic aspects of the protocol, and list only a few *TCG service provider interface (TSPI)* calls in the discussion. Some TSPI calls have

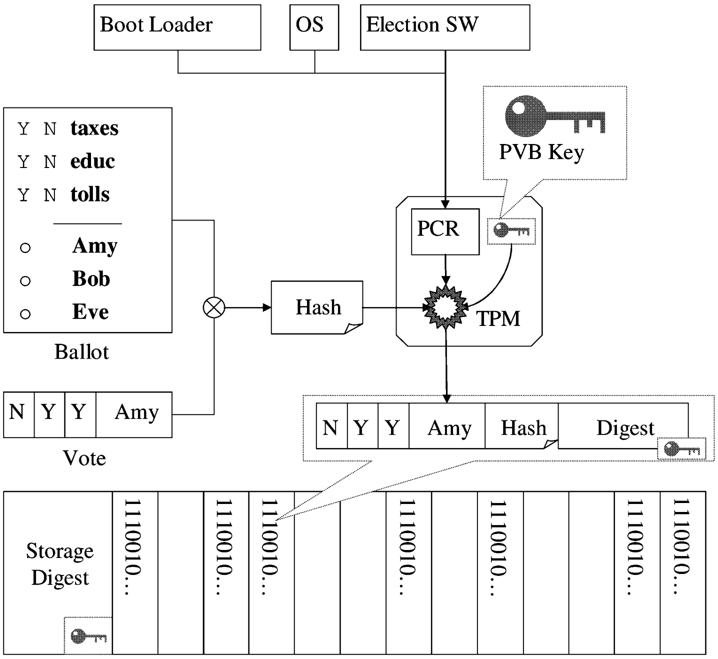

Fig. 2. Protocol loads the PVB key into the TPM to produce a signed digest of the vote and ballot. The PVB is usable only when the PCRs match a specified set of measurement values. The vote, digest, and hash are stored in a pseudorandom location on disk and the storage is signed whenever a new vote is inserted.

numerous arguments, and others require additional setup/takedown commands. We have purposely simplified these elements of the protocol to keep the presentation clear.

We assume the use of version 1.2 of the TPM specifications [36] and the associated TSPI implementation. Certain protocol features such as sealing are not available on prior versions of the TPM.

Our notation uses a lowercase h for hashing; P and S for public and private key operations, respectively, with a subscript indicating the key being used, and an inverse notation to indicate decryption using same. As an example:

$$\begin{aligned} A \rightarrow B :& P_B(P_A, B) \\ A \leftarrow B :& S_B(r_B, A) \\ Verif_B =& P_B^{-1}(r_B, A) \end{aligned}$$

A sends its public key and B's identity in a message encrypted with B's public key, B signs a message r_B to A (using B's private key), and A decrypts B's message using B's public key. Note the dual use of "key as a function" and "key as an entity."

B. Protocol Actors

Several actors and software items are referenced in the Protocol Steps section for convenience, and include the following:

1) **SWvote**—the platform software (voting application and operating system, plus any configuration files) needed for the polling phase;
2) **SWinit**—initialization software used during platform initialization;
3) **Platform**—the DRE computing unit, including the TPM, the user interface, CPU, memory, persistent storage (disk or other), I/O channels, and BIOS;
4) **Storage**—persistent storage on the DRE, e.g., hard disk.

C. Protocol Phases

The Platform Initialization and Platform Software Load and Key Creation steps are assumed to be conducted in a trusted environment.

Platform Initialization, Software Load, and Key Creation:

1. $TEA \rightarrow Platform$: **TakeOwnership**$(ownerPass, srkPass)$,
   $Platform \rightarrow Storage : meta_{SRK}$
2. $TEA \rightarrow Platform$: **CreateDelegation**$(SRK,$
   $\text{DELEGATE\_LoadKey}, srkPass, pollopenPass)$,
   $TEA \rightarrow Platform$: **CreateDelegation**$(Owner,$
   $\text{DELEGATE\_OwnerClear}, ownerPass, pollclosePass)$
3. $TEA \rightarrow Platform$: **Key_CreateKey**$(PVB, srkPass,$
   $pcrComposite)$,
   $Platform \rightarrow Storage : P_{SRK}(PVB), meta_{PVB}$
4. $Platform \rightarrow Storage : S_{PVB}(h(VoteStorage))$
5. $Platform \rightarrow TTA(\text{via } TEA) : P_{PVB}$
6. $TTA \rightarrow Storage : P_{TTA}$

1) The TPM is physically reset to erase any previous ownership. SWinit is installed and booted. The TEA chooses a password for the platform's TPM, *ownerPass*, and a password for the SRK, *srkPass*, and then invokes TPM_TakeOwnership to create the asymmetric SRK within the TPM. The SRK is protected by *srkPass*. The private portion of the SRK never leaves the TPM. The platform exports the SRK metadata, which includes information needed to reference the SRK later on, to platform persistent storage.
2) The TEA creates two delegations: a load key delegation, protected by *pollopenPass*, allowing the TPJ to use the SRK to load the PVB; and an ownership delegation, protected by *pollclosePass*, allowing the TPJ only to clear ownership in the Voting Termination step. These passwords are kept secret until election day. Delegation grants

only the needed rights to the TPJ without disclosing the full-use passwords, maintaining least privilege.

3) The TEA supplies expected software PCR measurements to SWinit which stores them in a PcrComposite object. SWinit calls Key_CreateKey with *srkPass* to create the PVB—the PVB is both a child of the SRK and bound to the PcrComposite values. The platform calls RegisterKey to store the encrypted PVB keypair and the metadata (called a *blob*) to platform persistent storage, wrapped by the public portion of the SRK.
4) A large area to store the votes called *VoteStorage* is allocated and initialized along with a separate area for cryptographic audit logs.[3] Storage consists of fixed-size slots, each initialized to a sentinel value (e.g., all zeroes). The storage should be generously sized to accommodate a large number of votes, and to reduce the probability of collisions when recording votes. The TEA signs the empty vote storage area with the PVB.
5) The TPM exports the public portion of the PVB to the TEA, and the TEA securely transmits the public key to the tallying authority. (PKI could create an integrity-protected channel between the TEA and TTA in this step, to ensure that the TTA receives the correct public key.)
6) The TTA's public key is installed on the platform to encrypt the storage during Voting Termination—Precinct.

Finally, the TEA installs SWvote onto the platform. The audit storage area is created, hashed, and signed. The platform is shut down, securely erasing the values of the volatile registers including the PCRs and any loaded keys or other authorization data. The platform is delivered to the precinct.

Election Day Initiation (and Reboot):

1. $TEA \rightarrow TPJ : \textit{pollopenPass}$
2. $TPJ \rightarrow Platform : \textbf{LoadKey}(PVB, \textit{pollopenPass})$
3. Verify $h(VoteStorage) = P^{-1}_{PVB} h(VoteStorage)$

1) The TEA reveals the delegation *pollopenPass* to the TPJ. This delegation password allows the TPJ only key loading, as opposed to unrestricted access to the SRK, and can be posted publicly.
2) The TPJ boots SWvote, causing measurements of SWvote to be extended into the TPM's PCRs. SWvote attempts to load the PVB key, which succeeds only if: a) *pollopenPass* is entered, and b) the PCRs match the measured PCR values of the certified SWvote software. If either condition fails, the PVB key cannot be used and the TPJ is alerted. Note that *pollopenPass* could be stored on the platform, allowing periodic reboots throughout the day to ensure a fresh set of PCR measurements.
3) The platform verifies the value and the signature on the storage area's recorded hash value. This ensures that that the storage (and audit log) is in a consistent and valid state, meaning that no votes have been improperly inserted, deleted, or modified. If the signed hash value is invalid, then the storage is corrupt and the administrator can be notified or the unit can be shut down. An audit entry is created reflecting the result of the boot. (After every entry, the audit log is signed securely.)

Voting and Recording:

1. $Voter \rightarrow Platform : vote$
2. $i \leftarrow \textbf{RANDOM}(1, \text{sizeof}(VoteStorage))$
3. $Platform \rightarrow VoteStorage[i] : vote, S_{PVB}(h(vote \parallel ballot))$,
$Platform \rightarrow Storage : S_{PVB}(h(VoteStorage))$

1) The voter receives an electronic ballot from the TPJ (possibly via voter registration card) and presents it to the platform. The platform displays the ballot to the voter, and the voter commits her choices to the SWvote software. (Aborts are possible on each race, or on the whole ballot, and are recorded in vote storage as such.)
2) A pseudorandom offset into the vote storage is computed, and adjusted for collisions. (The pseudorandom seed is obtained from the TPM which, in turn, is seeded with system randomness and TPJ randomness at platform reboot.)
3) The software hashes the vote and ballot data into a hash object using Hash_SetHashValue, then calls Hash_Sign to sign the hash inside the TPM with the PVB private key. Atomically, the vote record is inserted into storage and the storage area hash is updated.

The audit log is updated after every vote with any information required by the higher level protocol (but none that threatens voter privacy). This procedure is repeated for subsequent voters. As added security, the TPJ keeps counts of how many people attempted to vote and completed voting on each DRE.

Voting Termination—Precinct:

1. $TEA \rightarrow TPJ(\text{and } TTA) : pollclosePass$
2. $Platform \rightarrow TPJ : P_{TTA}(VoteStorage,$
$S_{PVB}(h(VoteStorage \parallel pollclosePass)), P_{PVB})$
3. $TPJ \rightarrow Platform : \textbf{OwnerClear}(pollclosePass)$

1) The TEA reveals *pollclosePass* to the TPJ and the TTA (used later), and can be posted publicly. The TPJ enters this in SWvote, witnessed by others.
2) The platform offloads the vote storage, the public PVB key, and a digest of the vote storage and the *pollclosePass* encrypted with the TTA's public key. (Omitting, or submitting an invalid *pollclosePass* in this step reveals premature precinct termination to the TTA.)
3) The TPJ clears ownership of the TPM. Hereafter, the PVB private key can never be used since the internal TPM state enabling its use has been erased. Cleared units can be rebooted and tested to validate that the PVB blob cannot be loaded.

The above events are audited and must be officially witnessed. The audit log is offloaded, but also retained on the platform. The encrypted data are transported to the Trusted Tallying Authority (TTA).

Tallying:

1. Decrypt : $P^{-1}_{TTA}(VoteStorage,$
$S_{PVB}(h(VoteStorage \parallel pollclosePass)), P_{PVB})$
2. Verify $h(VoteStorage \parallel pollclosePass)$
$= P^{-1}_{PVB}(h(VoteStorage \parallel pollclosePass))$
3. $\forall i \in \{1, 2, 3, \ldots\}$, Verify $h(VoteStorage[i])$
$= P^{-1}_{PVB}(h(VoteStorage[i]))$

[3] The audit log is not central to the use of the TPM, but is necessary for verification. All steps of the protocol should be logged. The Future Work section discusses important issues related to secure auditing and logging.

1) The TTA decrypts the transported data. The TTA looks up the supplied public PVB key against that supplied earlier by the TEA—if the key is not known, halt and report the error.
2) The TTA verifies the vote storage digest, and that it matches the *pollclosePass*. If verification fails, then either the precinct was terminated without knowledge of *pollclosePass*, or the storage digest is corrupt—halt and report an error.
3) The TTA verifies each recorded vote in the VoteStorage area—failure indicates a corrupt vote.

The TTA checks audit logs for proper sequences of operations, e.g., initiation, voting, and termination, as well as the proper signature on the audit logs. When satisfied with the votes and results, the TTA publishes vote digests and the public key of the PVB to the election trackers for public verification and adds the votes to the general tally.

Election Termination: At the conclusion of the election, the digital vote records and PVB public key are securely archived, allowing independent verification and historical analysis of the results.

D. Protocol and Implementation Enhancements

The protocol as described above has been kept simple for clarity, but certain design improvements could increase security and usability.

1) Authenticating DRE Presence—Preventing "Day-Of" Attacks: Day-of attacks are carried out by a malicious minority of trusted officials who might hide a valid DRE (perhaps intended as a spare) in a closet at the precinct to carry out a fake election. Policy could require some number of independent officials N to supply multiple passwords to terminate the precinct voting phase. When the TTA checks the vote storage, it also checks for knowledge of *pollclosePass* before considering the data to be legitimate. Dividing *pollclosePass* N ways prevents $N-1$ or fewer corrupt officials from slipping fake results into the tally. (In addition to election termination, N witnesses could be required to bring a machine into operation as well.)

2) Authenticating DRE Identity—Preventing Alternative Machine Substitution Attacks: A rogue official may attempt to substitute a terminal of his own choosing that exactly matches the software and configuration of authorized terminals in the polling booth. This attack may deceive the voter, compromising privacy; additionally, denial of service will occur because any votes collected by the machine will be signed by a PVB key unknown to the TTA, causing all such votes to be rejected. Our protocol could be extended to allow the voter, using third-party hardware, to verify the PVB signature on a voter-issued challenge to confirm platform correctness in the polling booth.

3) Other Enhancements: Assurance of the protocol requires that the storage remain in a consistent state, surviving simple power outages or even "pull-the-plug" attacks; implementations can utilize a commit/redo/undo protocol in a log-based recovery system for implementing stable storage [33]. Additional privacy can be provided by splitting the cast ballot and storing each voted race independently, defeating privacy attacks that might deduce relationships among different decisions, e.g., "most folks that voted for Amy voted 'no' to the tax hike." Repeat voting can be prevented by assigning a unique serial number to a voter, chosen from a large pool of random numbers each encrypted by the PVB public key. The PVB decrypts the supplied serial number, ensures membership in an authorized numbers table, and adds it to a list of used numbers before allowing the vote. Write-in candidates can be handled by a dynamic strings table referenced by the vote record, protected by encryption with the TTA public key.

VI. Security Arguments

We begin with a model of the adversary in terms of goals, capabilities, limitations, and information available for attack, and describe the security against several types of attacks given the model. Our adversarial model focuses on attacks against only integrity and privacy of the data storage and transmission mechanisms.

We assume that the adversary wants to subvert the election by changing the outcome or causing the public to question the validity of the election through modification, substitution, or unauthorized insertion of vote and ballot data. The adversary has physical access to the DRE unit before, during, and after the election, and can load software, reboot the platform, insert or remove data from local DRE storage. She can also insert software or otherwise change the behavior of the running election software without requiring reboot, perhaps using buffer overflows. The adversary also has access to secondary storage including the voter authorization card, and also the postelection precinct data bundled for transmission to the tallying location.

We assume that the adversary cannot recover the PVB private key. We assume that she cannot physically access the internals or influence the operations of the TPM without being detected.

A. Countered Attacks

Given this model, we describe high level attack vectors and show the infeasibility of the attacks given the adversary's limitations.

1) Ballot Modification and Misrepresentation: The attacker may attempt to modify ballot data in the voter authorization card to deceive the voter. Because the ballot is hashed with the vote and signed, the TTA would fail to verify the digital signature in the tallying phase, exposing the attack. If the attacker can somehow misrepresent the ballot on the screen, then the voter's choices would not reflect her intent, but this attack violates our assumption about certain trustworthy hardware.

2) Stored or Transmitted Vote Modification: If the attacker makes offline modifications to the individual vote—when the platform is turned off or when the vote is in transit—then the TTA would detect the difference on the vote and storage area digests when verifying the PVB signature (as recorded by the TPM), revealing the attack.

3) Stored or Transmitted Vote Insertion or Removal: The attacker may try to insert or remove votes from the DRE storage area. Since the TPM protects this storage area by signing a hash of the whole area by the PVB private key, the TTA would notice the integrity violation during verification of the storage digital signature.

4) In-Memory Data Modification: In-memory data modifications can occur if the attacker can subvert the correct operation of the software. Two methods of subversion include (1) *file injection attacks* that require a platform reboot to activate injected code (e.g., rootkits), and (2) *runtime integrity attacks* that alter the memory state of the running software without reboot (e.g., dormant activation flags, SQL injections, buffer overflows). Assuming (1), the PCRs would reflect measurements of the malicious software and invalid configuration files in the PCRs collected during boot, preventing the PVB from loading. Write-once storage can foil a vote storage replacement attack for case (2), an attack in which the subverted software replaces the vote storage area and commands the TPM to sign it. Although case (2) violates our assumption of correctly running software, defenses include write-only secure logging and dynamic runtime integrity measurement as discussed in Section VIII.

5) Observed Voter Order: An attacker might observe the polling place during the election and record the order of the voters using a DRE, and later correlate the stored vote order with his observations. Our protocol stores votes in a pseudorandom order onto the storage media, countering this attack against privacy by ensuring that the order of recorded votes is uncorrelated with the order of voter interactions.

6) Election Substitution/Day-Before Attack: One or more officials charged with safeguarding the machines activate and vote on properly initialized voting terminals the day before the actual election and then attempt to substitute the malevolent data at the end of the election. This attack is prevented because the system controls when the PVB key can sign data through a password revealed only on the day of the election. Since the TPM refuses to sign anything with the PVB key without proper authorization, binding the key to the election phase prevents the day-before attack. Further, password guessing attacks are detected and resisted by declining performance of the TPM, to the extent that certain TPM implementations will completely shut down once a failed authentication threshold is achieved.

B. Attacks Not Countered

Several classes of attack cannot be prevented by our protocol, or by any protocol that uses the TPM.

1) *Hardware Attacks:* These are specific hardware attacks that affect any complex software and hardware system:
   a) Memory attacks via rogue devices—devices could use *direct memory access (DMA)* to manipulate system memory during the vote casting process to display an incorrect ballot while recording a hash of the correct ballot. One partial defense is deactivating DMA on platforms, the other is physical security of ports; however, prior voting systems analyses show that locking down port access never fully solves the problem [18].
   b) Device tampering—misrepresenting a ballot to a voter can cause her to cast a vote that opposes her intention. Our protocol cryptographically binds the contents of the ballot with the vote, but it is difficult to prove what the voter actually saw when making her decision.[4]
2) *Other Attacks:* The following attack classes are not mitigated by the protocol: insider attacks, including coercion or payoff of a trusted entity; attacks against the higher level election system that uses our protocol; sophisticated physical attacks such as TPM power analysis, microscopy, or disassembly (easy to detect in the polling precinct); destruction of machines, resource exhaustion, and other denial of service attacks; procedural breakdowns where the TPJ fails in his duty allowing repeat DRE visits by the same voter; and overt physical tampering. Certain insider attacks could be mitigated by using shared secrets (for instance, defending against the day-of attack), but the remaining problems require correct procedural controls.

VII. Benefits and Limitations

The main benefit of the protocol is that trusted hardware assures the election authority of the integrity of the software and ballot data during voting, and the integrity of the vote data during storage and transmittal, increasing the security of the election. It also allows vote collection only during the legitimate election period. By delegating critical cryptographic operations to trusted hardware in a verifiable way, we can reduce risk and enjoy the usability benefits of DRE systems. Other benefits include that it:

1) is readily implemented with the TSPI;
2) works for any supported TPM and platform;
3) supports Static Core Root of Trust or Late Launch trust models—Late Launch is a special mode that ensures full measurement of the system components without trusting any of the software or firmware, but requires special CPU and chipset extensions such as Intel Trusted eXecution Technology (TXT).

One limitation of the protocol is its dependence on trusting the hardware. There are several respected authorities that validly argue that hardware is inherently opaque, and that any system (including ours) that delegates critical functionality to the correct operation of hardware is too risky. Further, some authorities struggle to accept foreign-made cryptographic hardware modules as trustworthy for processing sensitive national data such as elections.

Another limitation is that the PCR measurements verify only that the correct software is running, not that the software is running correctly. Validating correct software design and operation requires techniques such as formal proofs, trusted compilers, branch test coverage, and dynamic attestation of data structures [19]. Software-independent E2E systems also offer integrity protection against software faults.

Last, our present design provides no assurance to the voter that the machine is an authorized device or is configured or behaving correctly. As mentioned earlier, the protocol could be revised to include a challenge and verification step from the voter, but this raises scalability and accessibility concerns (e.g., can all voters obtain the necessary hardware?) and must be designed very carefully.

VIII. Future Work

This paper represents a start at using trusted hardware to mitigate some risks of DRE. One area related to the protocol is to modify the TPM specification to manage count-limited signing keys. This feature could allow numerous smaller vote storage areas—each signed with its own unique PVB—instead of one large one, to reduce the risk of total storage compromise. Sarmenta *et al.* [29] refers to this as *clobs*—count-limited objects. Another task would be to prove the protocol properties formally

[4] In the 1990s, Democracy Systems, Inc. offered a verification product called VoteGuard that recorded a video log of screen images presented to the voter.

to ensure that the security claims are satisfied under the stated assumptions.

A prototype could validate correct use of the TPM, and incorporate a full voting application to show usability with higher level voting systems. The protocol can be extended to include and improve E2E cryptographic audit trail technology. For instance, technologies such as Scantegrity [6], [7] empower each voter to verify that her vote was recorded and tabulated correctly, but verification takes place only after all results are reported. Our protocol can catch problems much sooner than is possible without technology assistance.

The problem of runtime integrity attacks—in our case, compromise of the live platform after the PVB has been loaded—can be addressed by at least two research areas. Policy-driven, secure write-once log storage could be used to verify historic system state and event occurrence cryptographically, preventing surreptitious wholesale replacement of the voting storage. The challenge is determining what data to record to properly balance the needs of voter privacy and public verifiability. Dynamic attestation of software state could thwart live software attacks by measuring the running system, perhaps with the help of virtualized environments, to verify the correct state of system memory structures [19]. Advances in both of these areas would benefit both electronic voting and information assurance in general.

IX. Conclusion

We have created a protocol based on hardware TPM enforcement of attested software state that resists vote modification, insertion, election replacement, and augmentation, and can reveal the use of incorrect software during the election data gathering phase. Our protocol works by protecting the integrity of both data at rest and data in transit as well as protecting voter privacy, and is compatible with higher level election techniques including end-to-end systems. We have shown in practical detail how trusted hardware can reduce the required trust base for electronic voting. Our work enables meaningfully more secure DRE voting with excellent usability and accessibility.

Acknowledgment

The authors would like to acknowledge D. Challener, H. Finney, L. Sarmenta, and J. Osborn for sharing their knowledge of the TPM; reviewers G. Walker, D. Heine, J. Land, D. Paulhamus, and the CSRG group of APL; and J. Pinkston of UMBC. The authors are grateful for the discussions and insight from D. Phatak, J. Krautheim, and members of the UMBC Cyber Defense Lab. Finally, they thank their IEEE reviewer who pointed out a remanence issue in an earlier draft.

References

[1] W. A. Arbaugh, "The real risk of digital voting?," *Computer*, vol. 37, no. 12, pp. 124–125, 2004.

[2] A. Aviv, P. Černy, S. Clark, E. Cronin, G. Shah, M. Sherr, and M. Blaze, "Security evaluation of ES&S voting machines and election management system," in *Proc. Conf. Electronic Voting Technology (EVT'08)*, Berkeley, CA, 2008, pp. 1–13, USENIX Association.

[3] D. Bowen, Top-to-bottom review Secretary of State of California Tech. Rep. 2007 [Online]. Available: http://www.sos.ca.gov/elections/elections_vsr.htm, Last accessed Jul. 31, 2009

[4] K. Butler, W. Enck, H. Hursti, S. McLaughlin, P. Traynor, and P. McDaniel, "Systemic issues in the Hart InterCivic and Premier voting systems: Reflections on Project Everest," in *Proc. Conf. Electronic Voting Technology (EVT'08)*, Berkeley, CA, 2008, pp. 1–14, USENIX Association.

[5] D. Challener, K. Yoder, R. Catherman, D. Safford, and L. van Doorn*, A Practical Guide to Trusted Computing*, 1st ed. Upper Saddle River, NJ: IBM Press, 2008, no. 978-0132398428.

[6] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, L. R. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman, "Scantegrity II: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes," in *Proc. Conf. Electronic Voting Technology (EVT'08)*, Berkeley, CA, 2008, pp. 1–13, USENIX Association.

[7] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. Sherman, and P. Vora, "Scantegrity: End-to-end voter-verifiable optical scan voting," *IEEE Security Privacy*, vol. 6, no. 3, pp. 40–46, May/Jun. 2008.

[8] Direct recording electronic (DRE) technical security assessment report State of Ohio Office of the Secretary of State, Tech. Rep., 2003 [Online]. Available: http://www.sos.state.oh.us/sos/hava/compuware112103.pdf, Compuware Corporation. Last accessed Mar. 15, 2008

[9] Decomisan varios ordenadores en la casa presidencial con los resultados de la consulta que queria hacer zelaya. (Rough Trans: Computers seized with bogus election results pre-loaded) Europa Press [Online]. Available: http://www.europapress.cat/internacional/noticia-decomisan-varios-ordenadores-casa-presidencial-resultados-consulta-queria-hacer-zelaya-20090717221327.html, Last accessed Jul. 31, 2009

[10] A. J. Feldman, J. A. Halderman, and E. W. Felten, "Security analysis of the Diebold AccuVote-TS voting machine," in *Proc. USENIX Workshop on Accurate Electronic Voting Technology (EVT'07)*, Berkeley, CA, 2007, USENIX Association.

[11] R. Fink and A. Sherman, "Combining end-to-end voting with trustworthy computing for greater trust, privacy, accessibility and usability (summary)," in *End-To-End Voting Systems Workshop*, Oct. 2009, National Institute of Standards and Technology.

[12] S. N. Goggin, M. D. Byrne, J. E. Gilbert, G. Rogers, and J. McClendon, "Comparing the auditability of optical scan, voter verified paper audit trail (VVPAT) and video (VVVAT) ballot systems," in *Proc. Conf. Electronic Voting Technology (EVT'08)*, Berkeley, CA, 2008, pp. 1–7, USENIX Association.

[13] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: Cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, no. 5, pp. 91–98, 2009.

[14] P. S. Herrnson, R. G. Niemi, M. J. Hanmer, and B. B. Bederson*, Voting Technology: The Not-So-Simple Act of Casting a Ballot*. Washington, DC: Brookings Institution Press, 2008, ISBN 0-8157-3563-4.

[15] H. Hursti, Diebold TSx evaluation: Critical security issues with Diebold TSx Black Box Voting, Renton, WA [Online]. Available: http://www.bbvdocs.org/reports/BBVreportIIunredacted.pdf, Last accessed Mar. 15, 2008

[16] A. R. Jorba, J. Antonio, O. Ruiz, and P. Brown, "Advanced security to enable trustworthy electronic voting," in *Proc. 3rd Eur. Conf. E-Government*, 2003, pp. 377–384.

[17] J. Kelsey, "Strategies for software attacks on voting machines," in *Developing an Analysis of Threats to Voting Systems*. Gaithersburg, MD: National Institute of Standards and Technology [Online]. Available: http://vote.nist.gov/threats/papers/stategies_for_software_attacks.pdf, Last accessed Mar. 15, 2008

[18] T. Kohno, A. Stubblefield, A. D. Rubin, and D. S. Wallach, "Analysis of an electronic voting system," in *IEEE Symp. Security and Privacy*, 2004, p. 27.

[19] P. A. Loscocco, P. W. Wilson, J. A. Pendergrass, and C. D. McDonell, "Linux kernel integrity measurement using contextual inspection," in *Proc. 2007 ACM Workshop On Scalable Trusted Computing (STC'07)*, New York, 2007, pp. 21–29, ACM.

[20] J. Miller and J. Krosnick, "The impact of candidate name order on election outcomes," *Public Opinion Quart.*, vol. 3, no. 62, pp. 291–330, 1998.

[21] L. Norden*, The Machinery of Democracy: Protecting Elections in an Electronic World*, ser. Voting Rights and Elections. New York: Brennan Center Task Force on Voting System Security, Brennan Center for Justice at NYU School of Law, 2006.

[22] N. Paul and A. S. Tanenbaum, "Trustworthy voting: From machine to system," *Computer*, vol. 42, no. 5, pp. 23–29, 2009.

[23] S. Pearson, *Trusted Computing Platforms: TCPA Technology in Context*. Upper Saddle River, NJ: Prentice-Hall, 2003, no. 978-0130092205.

[24] Trusted agent report: Diebold AccuVote-TS voting system State of Maryland General Assembly, Department of Legislative Services, Annapolis, MD [Online]. Available: http://www.raba.com/press/TA_Report_AccuVote.pdf, RABA Innovative Solution Cell (RiSC), Dr. Michael A. Wertheimer, Director. Jan. 2004. Last accessed Mar. 15, 2008

[25] T. Rössler, H. Leitold, and R. Posch, "E-voting: A scalable approach using XML and hardware security modules," in *IEEE Int. Conf. e-Technology, e-Commerce, and e-Services*, 2005, pp. 480–485, 2005.

[26] J. Rutkowska, Introducing Blue Pill [Online]. Available: http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html Last accessed Feb. 2009

[27] A. Sadeghi, M. Selhorst, C. Stüble, C. Wachsmann, and M. Winandy, "TCG inside?: A note on TPM specification compliance," in *Proc. First ACM Workshop on Scalable Trusted Computing (STC'06)*, New York, 2006, pp. 47–56, ACM.

[28] Risk assessment report: Diebold AccuVote-TS voting system and processes (unredacted) Science Applications International Corporation, State of Maryland Department of Budget and Management, Annapolis, MD, Sep. 2003 [Online]. Available: http://www.bradblog.com/?p=3731, Last accessed Mar. 15, 2008

[29] L. F. G. Sarmenta, M. van Dijk, C. W. O. Donnell, J. Rhodes, and S. Devadas, "Virtual monotonic counters and count-limited objects using a TPM without a trusted OS," in *Proc. First ACM Workshop on Scalable Trusted Computing (STC'06)*, New York, 2006, pp. 27–42, ACM.

[30] DriveTrust technology: A technical overview. Seagate Corporation, Seagate, Tech. Rep., 2006 [Online]. Available: http://www.seagate.com/docs/pdf/whitepaper/TP564_DriveTrust_Oct06.pdf, Last accessed Feb. 2009

[31] P. E. Sevinç, M. Strasser, and D. A. Basin, "Securing the distribution and storage of secrets with trusted platform modules," in *Proc. Workshop in Information Security Theory and Practices (WISTP)*, 2007, pp. 53–66.

[32] A. Sherman, "Election tracker: A new tool for greater election transparency," presented at the VoComp 2007, Jul. 2007, Rump session.

[33] A. Silberschatz, H. F. Korth, and S. Sudarshan*, Database System Concepts*. New York: McGraw-Hill, 2005, no. 0-07-295886-3.

[34] M. Strasser, A software-based TPM emulator for Linux Semesterarbeit, ETH Zurich, 2004 [Online]. Available: http://tpm-emulator.berlios.de/, Last accessed Apr. 2008

[35] The TrouSerS Software Library [Online]. Available: http://trousers.sourceforge.net Last accessed Apr. 2008

[36] *Trusted Computing Group*, TCG TPM Specification Version 1.2 Revision 103, 2008 [Online]. Available: https://www.trustedcomputinggroup.org/specs/TPM, Last accessed Mar. 2008

[37] K. P. Yee, "Building Reliable Voting Machine Software," Ph.D. dissertation, Berkeley, CA, 2007.

Russell A. Fink received the Bachelor's Degree in computer science from the University of Maryland, College Park, and the Master's Degree in computer systems management from the University of Maryland, University College. He is working toward the Ph.D. degree in computer science at the University of Maryland, Baltimore County (UMBC).

He is a senior staff member of the Johns Hopkins University/Applied Physics Laboratory, Laurel, MD, and a member of the UMBC Cyber Defense Lab. His research interests include network security, trusted computing, high assurance platforms, signals processing, and secure voting systems.

Alan T. Sherman received the Sc.B. degree in mathematics (*magna cum laude*) from Brown University, the S.M. degree in electrical engineering and computer science from MIT, and the Ph.D. degree in computer science from MIT studying under R. L. Rivest.

He is an Associate Professor of computer science at the University of Maryland, Baltimore County (UMBC) in the Computer Science and Electrical Engineering Department, Director of UMBC's Center for Information Security and Assurance, and a member of the National Center for the Study of Elections at UMBC. His main research interest is high-security voting systems. He has carried out research in election systems, algorithm design, cryptanalysis, theoretical foundations for cryptography, and applications of cryptography.

Dr. Sherman is also a private consultant performing security analyses, an editor for *Cryptologia*, and a member of Phi Beta Kappa and Sigma Xi.

Richard Carback was born in Baltimore, MD. He received the Master's Degree in computer science from University of Maryland, Baltimore County (UMBC) in May 2008. He is currently working toward the Ph.D. degree in computer science at UMBC.

He is a Research Associate for Convergent Technologies Incorporated, Baltimore, MD, supporting computer security development and training efforts. Previously, he has worked as a Research and Teaching Assistant at UMBC, and a Software Engineer at L-3 GSI, Inc. His research interests include end-to-end election systems, privacy enhancing technologies, virtual systems security, computer network operations, cryptology, and other topics in computer security and information assurance.