APPROVAL SHEET

Title of Thesis: Applying Trustworthy Computing to End-To-End Electronic Voting

Name of Candidate: Russell A. Fink
Doctor of Philosophy, 2010

Thesis and Abstract Approved: _____
Dr. Alan T. Sherman
Associate Professor
Department of Computer Science and
Electrical Engineering

Date Approved: December 6, 2010

CURRICULUM VITAE

Name: Russell A. Fink
Permanent Address: 8625 Tower Drive, Laurel, MD 20723 USA.
Degree and date to be conferred: Doctor of Philosophy, December, 2010.
Date of Birth: 1968.
Place of Birth: USA.
Secondary Education: Eleanor Roosevelt, Greenbelt, MD.
Collegiate institutions attended:

University of Maryland, Baltimore County, Ph.D. Computer Science, 2010.
University of Maryland, University College, M.S. Computer Systems Mgt, 1997.
University of Maryland, College Park, B.S. Computer Science, 1992.
Prince George's Community College, A.A. Engineering, 1991.

Major: Computer Science.
Minor:
Professional publications:

1. Fink, R. A Statistical Approach to Remote Physical Device Fingerprinting. In Military Communications Conference, 2007. MILCOM 2007. IEEE, pages 17. IEEE, 2008.

2. Fink, R. and Sherman, A.T. Combining end-to-end voting with trustworthy computing for greater privacy, trust, accessibility, and usability (summary). In Proceedings of the National Institutes of Technology (NIST) workshop on end-to-end voting systems, October 13-14 2009.

3. Fink, R., Sherman, A.T., and Carback, R. TPM Meets DRE: Reducing the Trust Base for Electronic Voting Using Trusted Platform Modules. IEEE Transactions on Security and Forensics, 4(4):628-637, 2009.

4. Kewley, D., Fink, R., Lowry, J., and Dean, M. Dynamic Approaches to Thwart Adversary Intelligence Gathering. Proceedings of the DARPA Information Survivability Conference and Exposition II (DISCEX-II), Anaheim, CA, June 12-14, 2001.

5. Fink, R. Reliability Modeling of Freely Available Internet-Distributed Software. Fifth Intl Software Metrics Symposium (Metrics 1998), Bethesda, MD, Nov 20-21, 1998.

6. U.S. Patent. Fink, R. A., et al. Passive Forensic Identification for Network Access Control of Computing Devices. APL Invention Disclosure 2612-SPL, filed at USPTO on October 7, 2009.

7. U.S. Patent. Kruus, P., Fink, R., James, C., and Akinyele, J. PWALL—A Collection Of Security Components Designed to Eliminate the Risk of External Network Attacks When Operating Within a Virtualized Environment. Provisional disclosure filed February 29, 2008 (APL Case number 2588-0305).

8. U.S. Patent. Fink, R.A. Method of Passive Forensic Identification of Networked TCP/IP Communication Endpoints. Provisional disclosure filed January 18, 2008, U.S. Provisional Patent 61/022,029 (APL Case number 2446).

9. U.S. Patent. Fink, R.A., Brannigan, M.A., Evans, S.A., and Ferguson, S.A. U.S. Patent 7,043,633, Method and apparatus for providing adaptive self-synchronized dynamic address translation, issued May 9, 2006.

10. U.S. Patent. Fink, R.A., Brannigan, M.A., and Ferguson, S.A. U.S. Patent 6,826,684, Method and apparatus for providing adaptive self-synchronized dynamic address translation, issued November 30, 2004.

11. U.S. Patent. Fink, R.A., Brannigan, M.A., Evans, S.A., and Almeida, A.M. U.S. Patent 6,363,071, Hardware Address Adaptation, issued March 26, 2002.

Professional positions held:

Johns Hopkins University / Applied Physics Laboratory. Senior Engineer. (2006–Present).

Private Consulting Firm. Senior Software Engineer. (2003–2006).

Network Associates Labs. Senior Software Engineer. (2001–2003).

BBN Technologies. Senior Software Engineer. (1998–2001).

Orbital Sciences Corporation. Senior Software Engineer. (1996–1998).

Lockheed Martin Space Mission Systems. Software Engineer. (1992–1996).

ABSTRACT

Title of Thesis: Applying Trustworthy Computing to End-To-End Electronic Voting

Russell A. Fink, Doctor of Philosophy, 2010

Thesis directed by: Dr. Alan T. Sherman, Associate Professor
Department of Computer Science and
Electrical Engineering

End-to-End (E2E) voting systems provide cryptographic proof that the voter's intention is captured, cast, and tallied correctly. While E2E systems guarantee integrity independent of software, most E2E systems rely on software to provide confidentiality, availability, authentication, and access control; thus, end-to-end integrity is not end-to-end security.

Trustworthy Computing (TC) improves the security of software systems significantly. The *Trusted Platform Module (TPM)* protects secrets and enforces security policy in a self-contained cryptographic co-processor. Systems use TPMs to allocate security requirements not to untrustworthy software, but to tamper-resistant hardware, enabling applications such as digital rights management and secure computing platforms.

Our research found that adding TC to voting systems is possible, practical, and enhances privacy even in E2E systems by managing election secrets inside trustworthy hardware. We produced 4 major results:

- Analysis of how TC can benefit E2E
- Design that adds TPMs to *Direct Recording Electronic (DRE)* voting systems, binding ballots and votes, with software state, and enforcing election day policy
- Design that enables voters to verify voting platform system integrity using common and inexpensive programmable smart cards
- Two designs that add trustworthy receipt printers to the Scantegrity E2E voting system adding usability, security, and enabling alternative voter interfaces

Applying Trustworthy Computing to End-To-End Electronic Voting

by

Russell A. Fink

Dissertation submitted to the Faculty of the Graduate School of
the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2010

For My Family—Christina, Ryan, and Katy

Bob, Marie

Russell, Martha

My Sisters

Timothy Nagel

In loving memory of Millie and Stan, Jewel and John, Mary and Esther,

whose brightly burning stars guided my every step

ACKNOWLEDGMENTS

As with any major work, there are too many people to thank, and too little space.

My highest gratitude goes to my beautiful wife, Christina, who encouraged me to start the program, held me to it, and helped me every step of the way. You have been patient beyond belief. Thanks to Ryan and Katy for your patience and understanding when dad was "busy busy." Thanks to all of you for being a sounding board when I got stuck, and offering helpful and practical suggestions.

I thank my advisor, Dr. Alan T. Sherman, for everything, especially for carefully reviewing my writing and always having time to talk.

Thank you, members of my committee, for the help and advice—Dr. Pinkston, who knew and counseled me from start to finish; Dr. Phatak, your enthusiasm, creativity and brilliance are unparalleled; Dr. Stephens, your careful review ensured my thoughts were organized and complete; and Dr. Challener, you are the brightest gem in the trusted computing field. Thanks for sending that e-mail message to the TCG back in '09. Thanks to members of the *UMBC Cyber Defense Lab (CDL)*, especially Rick Carback, John Krautheim, and Mike Oehler during my formative years at UMBC.

Thanks to my employer, *Johns Hopkins University / Applied Physics Laboratory (APL)*, and my supervisors for giving me the opportunity to complete this. Thanks to the many people that have reviewed copies of this paper or offered advice along the way, especially

Ginny Walker, Justin Osborn, and Ryan Gardner. Special mention to Robert Holder, Markus Dale and Christina Pikas for helping out a fellow grad student.

Last, I thank the following people, for encouraging me to attempt a Ph.D.: Dr. Jayanta Sircar, Dr. Carlo Broglio, Donna Gregg, Alan and Fran B., and Dr. Ralph Semmel.

Contents

List of Tables

List of Figures

List of Abbreviations

AIK Attestation Identity Key 116

APL Johns Hopkins University / Applied Physics Laboratory ii

API Application Programming Interface 23

BIOS Basic Input/Output System 116

CA Certificate Authority 134

CDL UMBC Cyber Defense Lab ii

CHA Challenge Authority 80

DMA Direct Memory Access 75

DRE Direct Recording Electronic 150

E2E End-to-End 149

EA Election Authority 121

EK Endorsement Key 125

HAVA Help America Vote Act 126

ITA Independent Testing Authority 89

LPC Low Pincount 27

MITM Man-In-The-Middle 34

NIST National Institute of Standards and Technology 6

NSA National Security Agency 84

PCOS Precinct Count Optical Scan 116

PCR Platform Configuration Register 115

PDA Personal Digital Assistant 123

PIN Personal Identification Number 84

PLA Platform Authority 89

PAP Policy and Procedures 126

PJ Precinct Judge 62

PVB Platform Vote Ballot 84

qrcode Two-Dimensional QR barcode 117

RSA Rivest Shamir Adleman 87

SHA Secure Hash Algorithm 25

SMM System Management Mode 83

SRK Storage Root Key 116

TA Tallying Authority 62

TC Trustworthy Computing 149

TCG Trusted Computing Group 110

TNC Trusted Network Connect 83

TPM Trusted Platform Module 150

TSPI TCG Service Provider Interface 90

TSS Trusted Software Stack 53

TXT Intel Trusted eXecution Technology 76

VCS Verification Codes Secret 124

VVPAT Voter Verifiable Paper Audit Trails 58

Your every voter, as surely as your chief magistrate, exercises a public trust.

Grover Cleveland, Inaugural Address
March 4, 1885

Chapter 1

Introduction

ELECTIONS are critical to maintaining the leadership of a free society. During elections, voter's intentions can be recorded exactly once; voters are hesitant to trust anyone with their vote; and the data must be simultaneously kept private, free from modification, and open to verification. Election officials must ensure that differently-abled voters are represented in elections, data records are preserved and available long after the election, and decisions can be rendered quickly. To meet these goals, voting systems must address competing principles of voter utility, information security, and ease of administration—as well as cost, ease of deployment, and reusability.

Municipalities have applied electronic and computer technology, including fully-electronic *Direct Recording Electronic (DRE)* voting systems such as touch screen systems, to record, transmit, and tally votes. Electronics help election authorities handle ever larger numbers of voters within declining fiscal budgets. Although electronic voting offers ease of use, convenient and nearly instantaneous tabulating, it has come under close scrutiny in the past decade because of exploits in the software and operating systems that make up a majority of the systems. These risks have been so severe that some leaders advocate

moving back toward paper-only systems—and paper-only problems. However, disabled voters cannot use paper systems without assistance, paper methods lead to problems such as confusing butterfly ballots and "hanging chads," and are nearly impossible to tabulate efficiently.

Recent research has produced *End-to-End (E2E)* cryptographic voting systems that guarantee election integrity through *software independence*, a property that ensures the outcome of the election is not influenced by the actions of software. More precisely, software independent systems guarantee with high probability that the voters and election authorities will detect improper changes to election data made by malicious or defective software. E2E systems achieve software independence by giving the voter a way to check that his vote is included in the final election tally, and enabling anyone to verify the correctness of the final tally without learning how any particular voter voted. However, we observe that end-to-end integrity does not imply end-to-end security; in particular, E2E does not guarantee privacy of the vote casting process or transmission of the results for tallying, but instead allocates these responsibilities to trusted officials and physical security procedures. Further, there are some software components such as accessibility marking interfaces for the disabled that some voters trust implicitly, limiting the extent of the software independence claims.

In this dissertation, we investigate augmenting E2E election systems with recent developments in *Trustworthy Computing (TC)*, including the *Trusted Platform Module (TPM)*—a dedicated, separate cryptographic coprocessor that safeguards vital election secrets from compromised platform software by storing them in tamper-resistant hardware. Our thesis is that we can significantly advance the state of the art in voting by applying TC to the systems architecture of E2E voting systems to achieve greater privacy, usability, and accessibility without weakening E2E's strong integrity properties. We will manage election secrets safely by applying TPMs to marking devices, printers, and scanners for both E2E and DRE

systems. We will attest platform software state to voters and independent third parties to prove that the correct software is in charge of the platforms and maintaining voter privacy. The result will couple E2E integrity with privacy, usability, and accessibility provided by electronic systems in a way that no present day optical scan, electronic, or E2E system can achieve.

1.1 Research Overview and Contributions

Our research demonstrates the practicality of adding TPMs to electronic and E2E voting systems to improve usability and accessibility while maintaining security. Specifically, we have produced designs that:

- integrate TPMs with DREs, cryptographically binding ballots, votes, and software state, and enforcing election day policy with trustworthy hardware
- enable voters to verify voting platform system integrity using ordinary smart cards
- couple a receipt printer to the Scantegrity E2E voting system in a way that is usable, secure, and enables alternative voter interfaces

These designs were inspired by the ways that TC can augment E2E systems, achieving benefits made possible by trustworthy hardware: protecting voter privacy, identifying problems early, reducing the trust placed in complex physical chains of custody, and ensuring central control of the voting equipment through password enforcement of cryptographic keys. These benefits are explained in detail in four foundational works.

1.1.1 Combining End-To-End Voting with Trustworthy Computing

We advocate combining TC techniques—including TPM protocols, application attestation, and reduced software footprints—with E2E technologies, to provide voter and universal verifiability while enhancing privacy, accessibility, and usability through trustworthy electronic systems. TC techniques can bolster E2E systems through ensuring the correct software is running and managing cryptographic keys securely, thereby enhancing privacy, deterring confusion, detecting problems sooner, and making possible high-assurance electronic accessibility interfaces including for the blind. We analyze specific benefits that TC can bring to E2E systems such as *Scantegrity with Invisible Ink (Scantegrity)*.

1.1.2 Reducing the Trust Base for Electronic Voting Using TPMs

We reduce the required trusted computing base for DRE voting machines with a design based on the TPM. 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 the correct password revealed on election day. Using the PVB with the TPM can reveal unauthorized software, ballot modifications, vote tampering and creation of fake election records early in the election process.

1.1.3 A Human Attestation Protocol—Bootstrapping Trust Using TPMs

We present a protocol that enables a voter to verify that he is interacting with a valid electronic voting machine using a simple smart card. The technique uses a standard and

inexpensive displayless smart card and a scratch-off sheet with hidden verification codes, both obtained from an independent third-party. A TPM on the voting machine provides attestation evidence to prove that the correct software has been loaded and that the machine is controlled by the *Election Authority (EA)*. Our protocol detects *Man-In-The-Middle (MITM)* "proxy attacks" where a corrupt election official violates voter privacy by installing a malicious machine that communicates with a hidden legitimate machine, forcing it to answer attestation challenges.

1.1.4 Trustworthy Receipt Printers for the Scantegrity Election System

We present two independent designs that add a confirmation number receipt printing capability to the *Scantegrity with Invisible Ink (Scantegrity)* E2E election system, improving voter usability and accessibility while maintaining privacy. We present two designs that use the TPM to implement a receipt printer in Scantegrity: the first is a stand-alone marked position image duplicator, and the second is a marked sense translator that requires state and a connection to the *Precinct Count Optical Scan (PCOS)* scanner, but that has the possibility of eliminating the need for invisible ink in the Scantegrity system.

1.2 Merit and Broad Impact

This research shows that TC can improve privacy of E2E systems, improve security and privacy for DRE systems, can catch problems in the polling location, and enables strong enforcement of election day policies including start and stop times of voting. Our work shows that TC approaches offer a sound basis for securing voting systems, including high-watermark E2E systems.

In a broader sense, improving security on E2E systems using TC will lead to greater acceptance of E2E overall. Our results set the stage for application to many other data collection and dissemination paradigms that rely on software for security and privacy, including healthcare, financial processing, and homeland security.

1.3 Collaboration and Authorship

This dissertation includes four technical chapters that were created as independent, publishable works. Russell Fink independently thought of applying TPMs to DRE voting, and was the first to create a detailed engineering protocol describing where to apply the TPM. Fink designed all of the protocols and use cases presented in this dissertation, vetting them with his committee and colleagues. He is the lead author on Chapters 4, 5, and 6.

Fink is the co-lead author of Chapter 7 with Richard T. Carback. Fink created the detailed designs and TPM protocols, and collaborated with Carback on the high level design, requirements, security arguments and background. The contents of Chapter 7 appear in both Fink's and Carback's dissertations with approval of their respective dissertation committees and the graduate school.

Other co-authorship and publication information includes:

- Combining End-To-End Voting with Trustworthy Computing (Chapter 4) was jointly written with Alan Sherman, and a short summary was published in the 2009 *National Institute of Standards and Technology (NIST)* Workshop on E2E Voting Systems
- Reducing the Trust Base for Electronic Voting Using TPMs (Chapter 5) is joint work between Sherman and Carback, and appeared in the December, 2009 IEEE Transactions on Security and Forensics, Special Issue on Electronic Voting

- A Human Attestation Protocol—Bootstrapping Trust Using TPMs (Chapter 6) was joint work with Challener and Sherman, and owes significant credit to Ryan Gardner for formative discussions
- Trustworthy Receipt Printers for the Scantegrity Election System (Chapter 7) is joint work with Carback and Sherman. These ideas, particularly the receipt printer, benefited greatly from the wisdom of Ronald Rivest

1.4 Organization

The dissertation is organized as follows. Chapter 2 gives a overview of voting, focused mainly on electronic systems, and discusses the risks to current voting systems. Chapter 3 describes an adversarial model common to the rest of our designs and protocols. Chapter 4 analyzes some security gaps of E2E voting, and describes specific changes that can be made to E2E systems to incorporate TC for better privacy and control. It is also presents the central arguments that support our thesis statement. Chapter 5 discusses a protocol for augmenting DREs with TPM and provides the basis for understanding how the TPM works in subsequent chapters. Chapter 6 describes a way that the voter can verify the booted software state of computer systems including voting platforms, and Chapter 7 combines the ideas of TC and E2E through designs for a verifiable receipt printer for Scantegrity for greater security and privacy. Chapter 8 concludes the work.

Trust, but Verify (Russian proverb)

Trust $\oplus$ Verify (Russ Fink)

Chapter 2

Background

In this chapter, we introduce the reader to the concepts used throughout the dissertation. We give an overview of voting, our perspective on risks of using electronics and software in modern voting systems, and the techniques we use to secure voting systems against software-related attacks.

2.1 Voting Overview

When electing their leaders, voters must be sure that their votes are cast as intended, collected as cast, counted as collected, and that data records are preserved for auditing and future analysis while maintaining the privacy of their choices. Election officials must ensure that differently-abled voters are represented in elections, data records are preserved and available long after the election, and decisions can be rendered quickly.

2.1.1 Goals and Requirements

Voting systems must address several principles to ensure that everyone can vote and that everyone's vote is counted:

- Voter utility— accessibility to disabled voters, voting usability, transparency, understandable marking and casting mechanisms, and detecting problems in the system close to the point of origin
- Information assurance—integrity, privacy, accuracy, authenticity, availability, repudiated choice, non-repudiated cast [88], and small trusted chain of custody (to minimize the attack opportunities)
- Ease of administration—auditability, ease of administration, efficiency, policy enforcement, total cost of ownership

Voter Utility

Voter utility means simply to provide a usable and efficient election system that is easy to use; does not require previous training in voting or computer systems; needs only basic cognitive abilities such as the ability to compare sequences of characters (called *strings*); and above all, ensures that procedures and equipment do not confuse the voter into voting against his will.

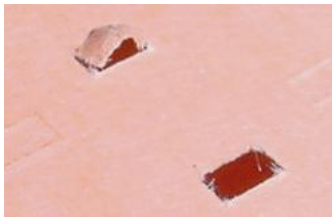
A voting system that is easy for the voter to use is said to be *usable*; a system that is easy for voters who are blind, or that have other disabilities, is said to be *accessible*. Special electronic interfaces provide accessibility in many ways—audible interfaces for the blind, large screen printing for poorly sighted voters, and sip-and-puff systems for paralyzed voters. If municipalities did not offer accessible systems, a disabled voter would have to trust a

human helper into the voting booth to mark, cast, and verify his ballot, thereby depriving his of the right to vote *in private*.[1]

Transparency is a part of voter utility, and is the property that enables the voter to understand what is going on with his vote and the election in general. A simple example is voting with colored marbles, as was done centuries ago. Each candidate in a *race*—an office or question, containing one or more possible candidates—is assigned a color, and each voter merely places the ball of the color corresponding to his choice into a hopper. Later, someone tallies up the colored balls and decides a winner. This process is transparent because it is obvious that the winner is determined by a simple plurality count. Unfortunately, "balls as ballots" is terrible for privacy, as others easily may observe what color ball is being dropped into the hopper. Further, each voter may be given balls of different sizes, so that the tallying officials know clearly how each person voted. Ballot *stuffing* is also possible, because corrupt officials may dump tons of extra marbles into the hoppers (or spill some out) after the public part of the election is complete. There are many other problems with this system, and readers are referred to Doug Jones' and others' treatment on the subject [53].

Voter utility also requires that voting systems provide an unbiased presentation of the ballot and an unquestionable capture of the voter's intent. An example of misrepresentation of ballots and voter confusion from the 2000 Presidential Election is described by Wand *et al.* in [100]. Among other problems were interpreting voters' preferences given failures in the voting technology. Figure 2.1 shows one such problem.

[1]We distinguish *voting* into three phases: *marking*, where the voter indicates his choice on the ballot; *casting*, where he commits his preferences to the official record; and *verifying*, where he ensures that his cast vote was tallied correctly. Most systems deployed today do not offer voter verification.



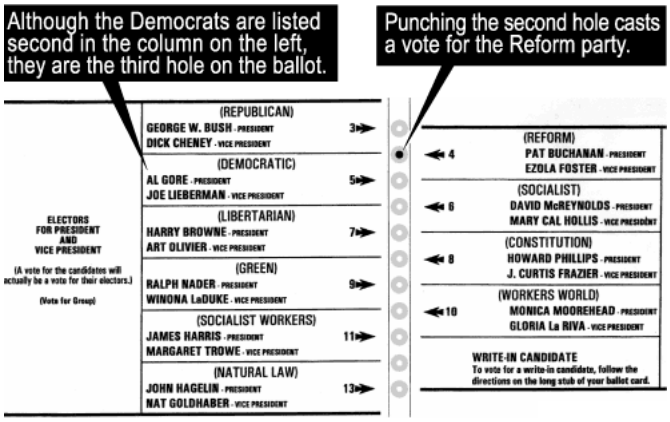


Figure 2.1: A *pregnant chad*, shown on the left, from the Computer Election Systems' Votomatic system. Is the pregnant chad an attempted vote, or a mistake that the voter caught before punching completely through? (Image from [53]) On the right is a sample butterfly ballot. Confused voters thought the second bubble indicated a choice for the second candidate on the left side of the ballot. Both technologies were used in Palm Beach County, Florida, during the 2000 U.S. Presidential election. (Image from [10])

Finally, catching problems early is important for utility so that the voter does not leave the polling place before his preferences are accurately recorded. Making the voter wait until after the election to verify his vote decreases the likelihood of him identifying problems, as too much time has gone by for him to remember all the details when he cast his vote.

Information Assurance

Information assurance is defined by some properties that must hold in order for information to remain correct. The basic properties are: *integrity*, ensuring that information is not changed by unauthorized parties; *privacy*, that sensitive information does not leak out; *authenticity*, that information was generated by known parties; *availability*, that information can be recorded and accessed when needed by authorized parties; and *non-repudiation*, ensuring that if a principal commits to some statement or fact, that this fact cannot be denied

plausibly by that principal later. We expect voting systems to provide all of these properties, or else the election could be subverted. For instance, if a voting system did not ensure integrity, then votes could be *flipped*—changed covertly—causing the wrong candidate to achieve office (*End-to-End (E2E)* systems offer strong integrity guarantees). Without privacy, voters could sell their votes or be coerced into voting for a specific candidate. Without authenticity, voters have no way of knowing that they are interacting with correct machines, and could be voting on an adversary's machine. Without availability, voting and tabulating systems are not available to support quick resolution of the election. If we allow voter repudiation, then anyone could initiate a *false challenge* of voting fraud, causing delays in the certification of election results.

Administration

Election systems must be easy to administer. Although ball and ballot systems may seem sufficient for very small numbers of voters, physical systems do not scale well. Imagine having to tabulate hundreds of tons of marbles in a presidential race! Good election systems capture votes in a way that is easy to tally and easy to verify. Total system cost impacts ease of administration, as well. Costs of producing the ballots, procuring and deploying computer systems, and training official poll workers are very high, and our officials are asked to conduct accurate elections in the face of declining budgets. As a result, a voting system that is considered "easy" to administer is one that is efficient, scalable, as well as affordable.

2.1.2 Architecture

A voting system has many components and phases. The *system architecture* describes components and their locations in the system; the human administrators, election judges,

and officials (called *actors*); and the phases of the election. Although real-world election system architectures vary from municipality to municipality, we establish a basic model to frame our designs.

Locations

Election officials install, configure, and perform preliminary tests of equipment in a *central election office*. This is also where they design and print ballots, and manage other materials required by voters and precinct officials. The *precinct polling location* is where the actual polling takes place. There are many precincts in a given election municipality, and therefore many polling locations. The *tallying location* is where officials count the precinct results and certify the election.

The polling place is staffed by election judges, with at least one judge from each competing political party, to ensure procedures are carried out correctly. The polling place is visited by officials prior to election day for equipment setup, and by voters on election day. It is a busy place with many different activities taking place, and can become confusing especially during peak periods of voter turnout.

Trust

Before continuing, we must distinguish the word *trust* from *trustworthy*. These two terms must not be used interchangeably. Trustworthiness is a characteristic granted to an entity after it has been observed to perform reliably in some capacity. Trust, on the other hand, is an assumption that is made independently from any prior performance. To trust a component, as defined by David Grawrock, is to rely on that component to perform an intended function for an intended purpose [43]. A component trusted to perform a given function is *expected* to perform that function, such that failure of that component likely means failure of the

system: failure of a trusted security component (including actor, role, or assumption) usually implies failure of system security.

Wise systems engineers strive to reduce the *trusted base* of systems as a result, minimizing the number and complexity of the things they trust to implement part of their security policy.

Verification supersedes trust. Verification is the process of confirming certain characteristics of an entity, for example through samples or measurements of behaviors or outputs, for the purpose of determining whether to interact with the entity. Verification is a process used by banks when you deposit a paycheck, for instance. Before crediting your account with the dollar amount of the check, the bank first verifies that your employer's account has the funds available. Trust, on the other hand, implies a reliance that defies verification. While the bank is busy verifying your employer's funds, you trust that your paycheck will clear so that you can pay your mortgage on time. If your employer lacks the funds, then your payment will be late. In this way, you trust your employer to pay you on time. The bank verifies your employer's account, but certainly your bank will not go out of business if your employer short-changes your wage. In security, verification is the goal, not trust; but where critical components cannot be verified, designers must settle for trust. Thus, the design goal of secure systems is to *minimize* the trust.

Trusted Roles and Procedures

Our notional system architecture assumes the following trusted roles:

***Election Authority (EA)*:** Responsible for correct execution of the election. Defines supporting roles, policies, procedures, and election rules. Defines requirements for election technology. Procures, configures software, and delivers election technology to precincts. Coordinates candidates and establishes the contests. Designs and creates

ballots, prints ballots, delivers to election place. Trains and certifies judges and minor officials. Resolves challenges, performs recounts as required. Archives election records according to municipal law.

***Precinct Judge (PJ)*:** Enforces election policies and procedures in the precincts. Sets up equipment in polling place. Resolves equipment failures and related polling place issues. Assists voters as required. Maintains privacy of results. Conducts parallel testing. Delivers results securely to tallying location.

***Tallying Authority (TA)*:** Tallies and publishes results, certifies the election and decides a winner. (The EA can perform these duties, but we assume distinct roles for generality.)

***Independent Testing Authority (ITA)*:** Reviews, compiles and tests election system software for the EA. Responsible for ensuring that the software meets the EA's requirements for the election system.

Parallel testing is an important procedure for election security. In parallel testing, election officials select sample machines at random from a pool of election-ready systems. Officials exercise each test system by voting on it using actual ballots, taking care to omit the test votes from the actual election tally. Parallel testing can catch certain software logic flaws and malicious code that may be designed to remain dormant until the day of the election, and then activate to flip votes, confuse voters, or inhibit the election in other ways. Although parallel testing is meant to catch corrupt software, Shamos [87] points out that the probability of randomly selecting a tampered unit is quite small. Clever malware can detect characteristic voter behavior, noting the difference between the "fist" of a high variation of voters versus a few select test individuals, or it could detect environmental characteristics using hardware-based techniques such as remote hardware fingerprinting to detect whether

it is in the polling place [57]. Still, it is useful against hardware failures or latent errors in software missed by ITA testing.

Phases

There are several phases in an election cycle, beyond just the day of the election. Table 2.1 shows a simplified representation of the election cycle. Note that with many phases, policy and procedure failures in early phases can propagate through to subsequent phases.

Phase	Activities	Time Before Election Day	Duration
Requirements	Define requirements for election systems according to local law	years	months
Procurement	Purchase and acquire equipment and software for voting	years	months
Independent Testing	Conduct testing of equipment. Perform clean compiles of the software	months	weeks
Software Installation	Load software onto voting platforms	months	days
Ballot Preparation	Prepare ballot designs, print ballots	weeks	days
Training	Hire and train poll workers on technology	weeks	days
Equipment Deployment	Deploy equipment, ballots, forms to precinct officials. Voting machines often stay overnight at election officials' houses awaiting deployment to the polling places	weeks	days
Precinct Setup	Install equipment at precinct.	1 day	hours
Election Day	Open the polling location, take zero counts of machines, activate machines, allow voting, close election at appropriate time. During voting, conduct parallel testing	(election day)	1 day
Results Tabulation	Send results to central location for tallying. Data are often transported by local law enforcement	1 day after	hours
Election Certification	Certify the results and announce a winner	days after	1 or 2 weeks, barring challenges

Table 2.1: Phases of the election cycle. Many steps take significant effort, and errors in early phases can influence later phases adversely.

Figure 2.2 is a notional architecture diagram showing the major election components and phases for a *Direct Recording Electronic (DRE)* election system.

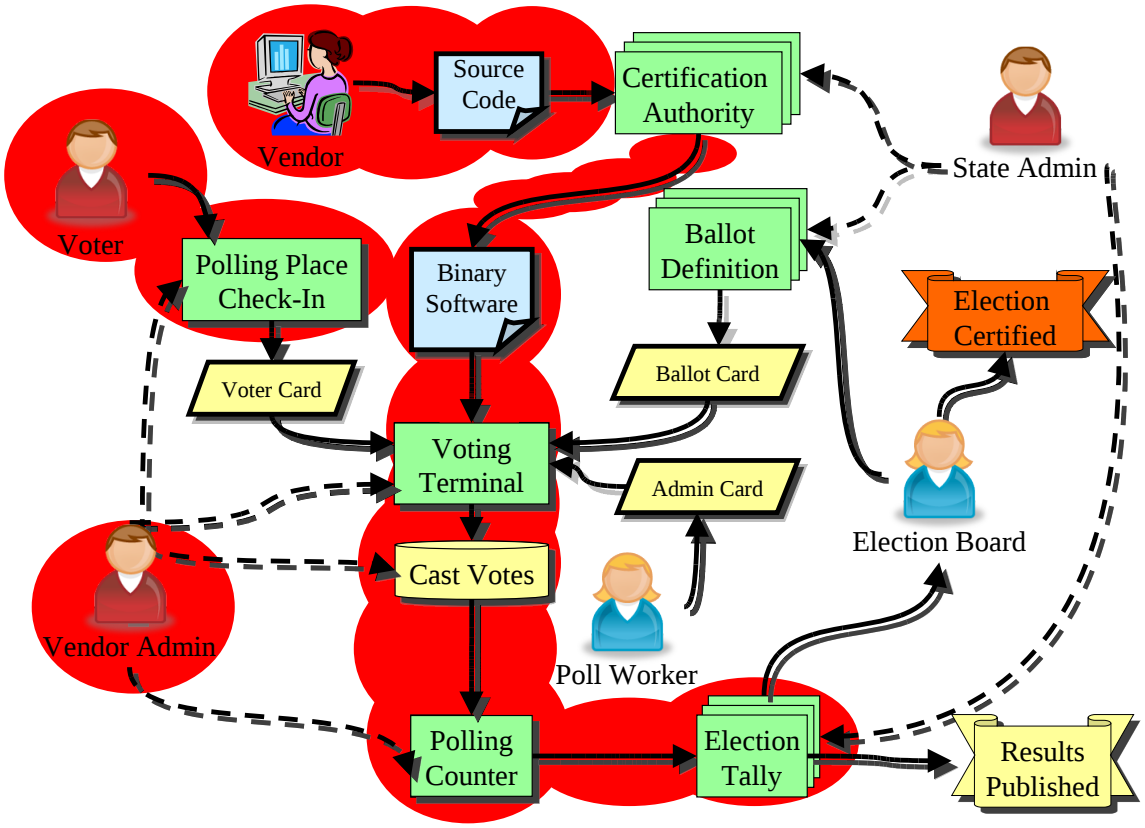


Figure 2.2: Notional architecture of a DRE election system. The highlighting indicates trusted components and paths, plentiful throughout the system.

2.1.3 Types of System

Many voting systems have been proposed to meet the goals of information assurance, ease of administration, and voter utility. We consider the ability of three contemporary voting technologies to solve problems: central-count paper record systems, electronic and specifically DRE systems, and E2E voter and universally verifiable systems.

Optical Scan

An optical scan system tallies votes by sensing marks that the voter makes on a paper ballot. Two types of optical scan systems include central count optical scan and *Precinct Count Optical Scan (PCOS)*. Both types scan a paper ballot, marked in various ways by a pencil, dark pen, or punched holes depending on the system, and translate these marks into choices in the contests. Central count and precinct count differ in where the paper records are scanned and processed. Central-count paper record systems implement a ballot-box style of voting where voters mark their ballot cards and drop them into a ballot box. At the close of the election polls, trusted officials seal and transport the boxes to a central location for scanning and tabulating. PCOS systems tabulate the records in the polling location, and officials record counts for each scanner at the close of the polls. Officials also seal the marked paper PCOS ballots and transport them to the EA for safe keeping in case a manual recount is required.

The benefits of optical scan systems are relatively inexpensive ballots, cheap marking systems, fairly basic scanning technology, and ballot records that the voter marks directly. Additionally, the voter can notate write-in candidates most easily on paper records.

Unfortunately, paper systems have many problems. They have large, trusted chains of custody to protect the privacy and integrity of the ballots and digital counts during transport from the precinct to the election authority. Paper systems are hard for blind or disabled voters to use, requiring special automated marking systems that may require assistance or training to use. Furthermore, the security of some automated marking systems is an afterthought, and privacy and integrity could be compromised for blind voters. Undervotes and overvotes—no mark and too many marks on a given race, respectively—are not detected by the paper ballot, and tabulating machinery can destroy paper records.

These and related problems with paper records have led to the creation of other systems.

Electronic and DRE

Electronic systems have been created to address usability problems, improve counting efficiency, and increase privacy. DRE systems fully automate the ballot presentation and data capture steps, reducing the time needed to scan and count the votes enabling quick election results. DRE provides much better usability and accessibility to voters than do paper records because the display and entry apparatus can accommodate voters with different abilities through high-contrast displays, audible ballots, multi-language ballots, and sip-and-puff systems for paralyzed voters. Electronic systems also detect undervotes and overvotes prior to vote casting.

Unfortunately, recent electronic and DRE systems implementations were poorly designed and are susceptible to undetected software replacement and injection attacks [32, 58]. Previous studies on security and implementation problems of current DRE systems include Kelsey's [55] catalog of DRE attack strategies, a threat analysis derived from attack trees by the Brennan Center [66], and research analyses by Kohno et al. [58], SAIC [77], RABA [71], and Compuware Corporation [27] on flawed commercial DRE implementations. Additionally, Hursti [49] analyzed the problems of unauthenticated software installs, and Feldman, *et al*. [32] analyzed the damage caused by viruses when policies and procedures are not followed. Additional vulnerabilities in modern DREs were uncovered in the EVEREST Project [15] and in the California Top-To-Bottom Review [14].

E2E

E2E election systems use voter verifiable cryptography to reveal data discrepancies. One E2E system is *Scantegrity with Invisible Ink (Scantegrity)* [23]. Scantegrity uses a single ballot sheet preprinted with ovals, corresponding to candidates, that contain invisible secret

codes. To vote, the voter uses a special ink pen to mark her choice on the ballot corresponding to her choice, revealing an alphanumeric code that she can check against results posted on a public server. The marked ballot is scanned with traditional PCOS scanners to record the voter's intent. Scantegrity is voter verifiable—the voter can verify that her vote was accurately captured using the revealed codes—and also universally verifiable, meaning that anyone can verify that the computation of the final tally is consistent with cryptographic audit logs. Besides implementing E2E, an additional strength of Scantegrity is that off-the-shelf PCOS scanners can interpret the ballot marks.

Figure 2.3 shows an example Scantegrity ballot.

Figure 2.3: Example Scantegrity ballot. Special verification codes are printed onto the ballot with invisible ink. Voters reveal the codes by marking the bubbles with a special pen, and verify these codes against the public bulletin board after the election. Standard PCOS scanners will read the marked Scantegrity ballot. (Image from [81])

2.2 Voting System Risks

Voting architectures are complex and involve many parts that must work correctly to ensure that every vote is recorded, cast, and tallied correctly. Component and interconnection complexity creates specific types of risks that can lead to attacks on system integrity and on voter privacy. Some of the risks are:

Procedures: Procedures must be carried out by many different election officials. Often, officials are not trained adequately or are inexperienced with managing elections. Election day, in particular, involves many specific procedures—from recording *zero tapes* at the beginning of the election, to ensuring that the hours of the polling location are followed (see [41] for a violation of polling location hours). Further, voting booth rules must be followed, including preventing voters from using cell phones in voting booths *e.g.*, to communicate to a vote buyer. Many times, not enough officials are on hand to handle the full voter load, leading to breakdowns in procedures.

Custody Chains: A chain of custody is a trusted path in which items are transported from one secure place to another. An example is moving printed ballots from the central authority to the precinct polling sites. An important chain of custody is the transport of precinct results to the central tallying location at the close of the election; failure for some votes to appear can lead to widespread disputes of the entire election.

Software Testing: Bev Harris reports that many ITAs are ineffective, staffed by inexperienced people or are rushed in an effort to certify a product quickly. The vendor employs the ITA, not the EA, so any bugs that are reported are free to be ignored [45]. Sadly, election authorities lack the expertise to do the testing, and must rely on the

results of the ITA and the honesty of the vendors to help certify correctness of the election equipment.

System and Software Complexity: Computer technology is complicated. In particular, system and application software is complex, is produced by developers of unknown pedigree, and is something not easily understood or inspected by the ITA. Voting software often relies on third-party libraries (*e.g.*, Microsoft Windows *Application Programming Interface (API)* and libraries) for which no source code is available, introducing complex interdependencies and latent errors that appear sporadically and may be hard to trigger in a cleanroom test lab environment. The Brennan Center of Justice performed a comprehensive assessment of voting technology in 2006, and discovered specific software risks that include software *injection* attacks—where malicious software is loaded onto the voting platform even during the election; *lifecycle* attacks, where malware is inserted during system and component libraries development; and misconfigurations and accidental networking, allowing attackers to compromise system software remotely. Further, the Brennan Center points out that software attacks are the least detectable, and most pervasive, of all possible attacks on election systems because they require the fewest "informed participants." [65].

These risks can be exploited by a clever and concerted adversary to tamper with the election results, hinder the certification of the winners, and call the entire voting system into question shaking public confidence in the system. The adversary receives a special treatment in our attack model described in Chapter 3.

2.3 Cryptographic Security

Before discussing the adversary model and attacks, we review the tools and technologies of *Trustworthy Computing (TC)* that we will use to defend against the various attacks.

2.3.1 Cryptography Primer

Cryptography is a field of mathematics that is used in data security. It concerns transformations of numbers through functions in such a way that ties transformations to pieces of information called *keys*. Data are secured by using keys with *encryption algorithms*—mathematical operations that transform data from a *plaintext*, or original representation, into a *ciphertext*, or transformed representation that bears no resemblance to the original (with high probability).

There are two types of encryption algorithms: *symmetric* algorithms, that use the same key to encrypt and decrypt data (also called *shared key* algorithms); and *asymmetric* algorithms that use one key to encrypt data and a separate but unique key to decrypt data (also called *public key* algorithms). In public key algorithms, the two keys are paired mathematically such that every key has a unique partner key. Keys such as this are called *key pairs*, where one key is called the *private* or *secret* key and its paired mate is called the *public* key. In practice, the public key can be shared freely, while the private key is kept extremely secure, often protected by a password and sometimes by a special piece of hardware like a smart card or cryptographic *token* or *fob*. The two keys can provide *confidentiality* and *authenticity* depending on how they are used.

A didactic example involving two communicating parties, Annie and Bob, illustrates the different use of keys. If Annie wants to send a private message $\mathcal{M}$ to Bob such that only Bob can read it, Annie encrypts $\mathcal{M}$ with Bob's *public* key. Bob decrypts $\mathcal{M}$ with his private

key. If Bob is the only one with access to the corresponding private key, then only Bob may decrypt and read the message, and message $\mathcal{M}$ was sent in confidence from Annie to Bob. This is denoted by $A \rightarrow B : P_B[\mathcal{M}]$.

Bob may wish to send an acknowledgment to Annie that he got her message. But, he may want to ensure that Annie knows it came from him, and not somebody else. Bob encrypts his reply $\mathcal{R}$ with his private key, and sends it to Annie. Annie attempts to decrypt the message using Bob's public key; if the decryption succeeds—that is, the message appears to be a properly formatted acknowledgment message, and not gibberish—then Annie knows $\mathcal{R}$ came from bob if she assumes that only Bob has Bob's private key. By using his private key, Bob is said to *sign* the message. This is denoted by $A \leftarrow B : S_B[\mathcal{R}]$. Signatures factor prominently in our dissertation.

Besides encryption, *hash algorithms* are used to implement non-invertible mathematical transformations on data. While encryption functions transform plaintexts into ciphertexts of at least the same size, hash functions can take arbitrarily large plaintexts as input and produce small, fixed-sized digital representations as output. These representations are sometimes called *thumbprints*, similar to how an inked thumbprint is a very compact representation of the thumb owner's identity. Hash functions have the property that for any two texts Φ and Υ, $hash(\Phi) = hash(\Upsilon)$ only when $\Phi = \Upsilon$ with high probability. In other words, a good hash algorithm resists *collision*, a condition where two separate texts hash to the same value. Strong algorithms make it computationally time-consuming for an adversary to find a colliding text given nothing more than some fixed input message. One popular hash algorithm is *Secure Hash Algorithm (SHA)*-1, which produces a 160-bit hash code for texts of arbitrary size, with a low probability of collision.

Because of these properties, hash algorithms can help indicate message integrity: if one bit in a message is changed, its hash value will change also. But what protects the

hash? If we use public key cryptography, we can sign the hash of a message to prove both authenticity of the sender, and that the message (and its hash) has not changed in transit. Signed hashes are called *digests*, and are used typically on e-mail messages, over secure connections, or anywhere that data can be intercepted and changed. Here is the full notation of how Annie verifies Bob's message:[2]

Part 1 - Annie Verifies Bob's Message

$$B : d_B = S_B[\textbf{Hash}(\mathcal{R})] \tag{1.1}$$

$$A \leftarrow B : \mathcal{R},\ d_B,\ \text{“}Bob\text{”} \tag{1.2}$$

$$A : annieHash = \textbf{Hash}(\mathcal{R}), \tag{1.3}$$

$$P_B = \textbf{GetPublicKey}(\text{“}Bob\text{”})$$

$$A : bobHash = P_B^{-1}(d_B) \tag{1.4}$$

$$A : (annieHash == bobHash) \longrightarrow \text{message valid, came from Bob} \tag{1.5}$$

1. Bob hashes his acknowledgment message and signs it to produce digest d_B.

2. Annie receives the message from Bob that includes the acknowledgment, the digest, and a claim that the reply came from "Bob".

3. Annie performs her own hash of the reply. She looks up Bob's public key (if she did not have it already).

4. Annie decrypts Bob's digest to get Bob's hash of the message.

5. Annie compares her own hash to Bob's decrypted hash. If they match, she knows the message is valid and came from Bob. (If they do not match, then either the message was changed in transit, or the message did not come from Bob.)

[2]Our protocols are written as major *parts* that each consist of separate *steps*. The numbered sentences that follow each part correspond to the steps within that part.

2.3.2 *Trusted Platform Modules (TPMs)*

The TPM is an embedded cryptographic processor and nonvolatile storage device meeting specifications provided by the Trusted Computing Group [97]. 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, monotonic counters, and the SHA-1 within tamper-resistant hardware.[3] 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 [97].

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 *Platform Vote Ballot (PVB)* is created as a child of the SRK. As additional protection, the TPM can enforce passwords on blobs to prevent unauthorized loading.

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

Ownership is enforced by an owner password that allows the administrator 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 administrator 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 *Basic Input/Output System (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 a type of TPM memory called a *Platform Configuration Register (PCR)*, a special volatile memory location 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 non-zero 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 to load the key only when the software measurements stored in the PCRs are correct.

Figure 2.4 shows the major functional components of a TPM.

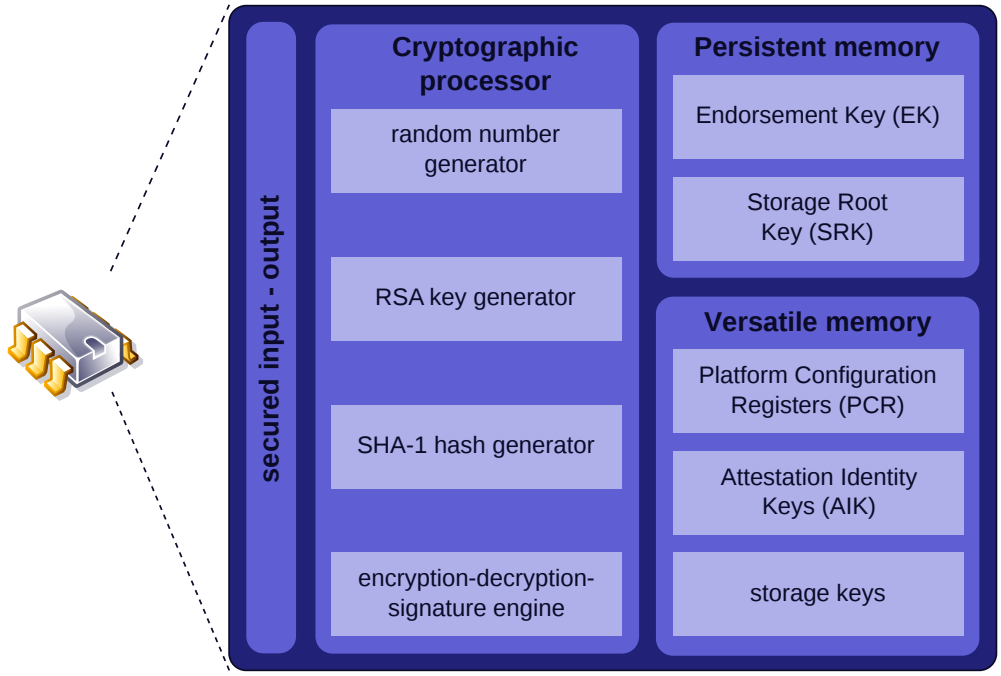


Figure 2.4: Functional block diagram of a TPM showing its major components. (Image credit: Guillaume Piolle)

2.3.3 Timing

Computers are built from semi-conducting electronic components with electrical properties that vary based on physical environment. Clock circuits, in particular, are subject to timing variations based on thermal conditions. Because clock circuits emit periodic pulses that let CPUs move from instruction to instruction, variations in temperature can cause CPUs—including that in the TPM—to function slightly faster or slower than normal. Moreover, each circuit is a unique, physical instantiation of a design, meaning that two implementations of the same design may exhibit different timing characteristics. Research has exploited these timing differences to identify specific computing platforms based on the rate of time gained or lost relative to a measurer [33, 57]. We suggest that hardware timing techniques can be

used to determine timing variance on a TPM's operations, such that we can tell how many components lie on the path between a challenger and a TPM. This idea is examined further in Chapter 6.

2.3.4 Security Analysis

To show the worth of our designs, we use *security analysis* to show attacks that we can defeat and, importantly, to justify the assumptions we make. An informal security analysis begins with a model of the attacker: goals, capabilities, and locations of the attacker; followed by the system assumptions and an examination of (a) how the design defeats the attacks, and (b) what can go wrong if some of the assumptions are relaxed or broken.

Indian EVMs [Electronic Voting Machines] are fully tamper-proof when used under complete administrative safeguards prescribed by the ECI.

Alok Shukla, Deputy, Election Commission of India (ECI), EVT/WOTE, August, 2010

Chapter 3

Adversary Model

We briefly define the adversary in terms of goals, capabilities, and possible attacks. We assume the same basic adversary model throughout the dissertation, so we discuss it separately before we present the designs.

3.1 Adversary Capabilities

Our adversary is technologically capable, and computer savvy. He is funded at a fairly high level, but is risk averse and wants to avoid detection (*e.g.*, the adversary may be an *insider*, and wants to remain employed for subsequent elections). Our adversary wants to subvert the election by changing its outcome wtihout detection. Barring an easy change of outcome, the adversary will settle for delaying its certification.

The adversary also wishes to establish continued presence in the election system, or cause election officials to consider reverting back to simpler, more easily compromised election systems benefiting the adversary. He has physical access to election systems and artifacts (*e.g.*, printed ballots) before, during and after the election, and can load software, reboot any

computer components, and insert or remove data from local storage. The adversary also has access to secondary storage including the voter authorization card, and also the post-election precinct data bundled for transmission to the tallying location.

The adversary knows how to take control of system software and its underlying storage. The adversary can make software changes before or during the election, even to machines in the polling booth.

3.2 Attack Classes

Several attack types are available to the adversary to help him achieve his goals. These attacks are separated from attack techniques, as one or more techniques may help carry out a particular type of attack.

3.2.1 Data and Presentation Manipulation

The attacker can modify data and its presentation. Data modification is the most basic form of attack against election systems, and it results in flipping enough votes tightening the margin of victory, causing an expensive manual tabulation of mail-in votes or even a hand-recount of all cast votes. Data insertion (ballot stuffing) and data deletion also are possible.

The adversary can attack data at rest, including data stored on local precinct devices, and can attack data in transit by subverting the trusted chain of custody in transporting precinct results to the central tallying location.

Subliminal attacks impact the voter and may cause his to become confused, change her vote, or not vote in a particular contest. These attacks occur when the adversary makes

subtle changes to the display, the font, or the order of the candidates on *Direct Recording Electronic (DRE)* user interfaces or on the ballot.

The attacker can change the way the ballot is presented through many means. The attacker may attempt to modify ballot data, either by reprinting physical ballots with incorrect or reordered information, or by modifying the data image stored in DRE voter authorization smart cards. The attacker may try to misrepresent the ballot on the screens in DRE systems, and may try to realign the touch screen areas in DRE systems causing voters to select the wrong candidates inadvertently.

3.2.2 Privacy

Privacy attacks are possible against the voters in the system. We distinguish two types—uninformed attack, where the voter is unaware of the attack; and informed attack, where the voter is a willing participant in disclosing his privacy for personal gain.

Uninformed privacy attacks include election queue observations, where an attacker might observe the polling place during the election and note the order of the voters that visit a polling booth. Later, the attacker can correlate the stored or scanned vote order with his observations to determine how a voter voted.

Informed attacks occur when the voter works in conjunction with the attacker. These attacks can be thought of dually as coercion attacks or vote buying attacks. In these attacks, voters carry out instructions on behalf of the attacker, sometimes communicating in real time with the attacker, transmitting display elements or prompts that the voter sees allowing the attacker to decide the course of action. For non-DRE systems, the voter can record sentinel marks on the ballot indicating his identity. The voter typically carries out some proof that he followed the adversary's orders, in order to complete the coercion / vote buying transaction.

3.2.3 Procedural

Attacks may occur against established election rules or procedures. An example is the *day-before* attack, where one or more officials charged with safeguarding voting machines activate them and vote on them the day before the real election and then substitute the malevolent data after the polls close. This attack is possible because many municipalities allow the voting machines to "sleep over" at election officials' houses the day before the election, stored in uncontrolled environments. Additionally, post-election attacks may be carried out by malicious poll workers keeping the polls open "just a little longer," violating election law but carrying out their will [41].

If poll workers are unobservant, attacks such as *chain voting* can occur, in which one cooperative voter carries a clean ballot out of the booth. The adversary marks the ballot with his choices, and then hands the ballot to a willing or confused voter entering the polling booth with instructions to cast the marked ballot and return with another clean ballot, perpetuating the cycle.

Other polling place attacks include *behind-the-curtain* attacks, where a legitimate polling station is secreted away immediately after poll opening and voted on repeatedly by attackers. Data from these machines are stealthily shuffled in with the legitimate polling data before transmitting to the central tallying location.

There are also machine substitution and *Man-In-The-Middle (MITM)* attacks in which an adversary replaces a legitimate election terminal with his own, or connects a legitimate election terminal to his own terminal. There is a related attack that can be described as a *hostage* or *kidnapped oracle* attack, where the attacker removes a legitimate terminal, replaces it with his own machine and connects his machine to the legitimate machine; during the election, any sort of integrity challenge sent to the attacker machine is relayed to the

hostage for response, fooling the voter into thinking he is interacting with a legitimate terminal when, in fact, he is not. Figure 3.1 illustrates such a case.

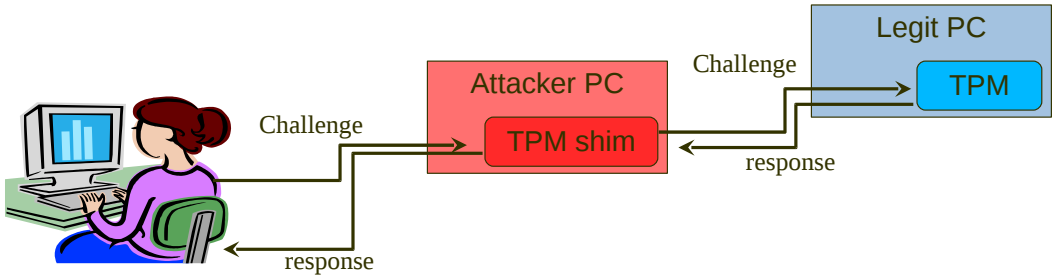


Figure 3.1: Example of the oracle / hostage attack. The voter on the left believes she is interacting with a legitimate terminal, because the verification challenge has been answered correctly by the hostage.

3.2.4 Discreditation

If the attacker is having trouble with a secure election system, he may take actions that raise a public question of the validity of the election through modification, substitution, or unauthorized insertion of vote and ballot data. If he can shake public confidence in the election system, then the adversary might sway election officials into reverting back to simpler election methods that are more easily compromised by the adversary.

3.3 Attack Techniques

Adversaries practice a number of attack techniques to disrupt the election process, violate privacy, or discredit the public's perception of the election system.

3.3.1 Software Attacks

Adversaries exploit the risks inherent in complex election systems. Attacks against system software come in two varieties—persistent attacks, and non-persistent attacks. Persistent attacks compromise software images on storage media for various computing platforms. By simply adding, changing, or removing an application or operating system component or shared library, the adversary can change the behavior of the application. One type of persistent attack is a *root kit*, in which malware virtualizes the installed software base and runs the system, altering the memory state of the election system software per the adversary's request. Non-persistent attacks, otherwise called live, software injection attacks, are more complex, and involve exploiting some programming errors (*e.g.*, array bounds violations on fixed memory buffers) to cause a crash and install the attacker's code. These attacks insert attacker software or otherwise change the behavior of the running election software without requiring reboot.

3.3.2 Other Attacks

Other attacks exist, but are beyond the focus of our work. They include human factors, a majority of corrupt officials, denials of service, and physical attacks. For instance, a single leader responsible for the elections that also happens to be up for re-election can manipulate the tally any way he wants, *e.g.*, in [29].

3.4 Assumptions

We make some basic security assumptions about systems that implement our designs. At a high level, they include assumptions on the system and trusted roles.

3.4.1 Security Assumptions

- Asymmetric cryptographic algorithms, including digital signature keys and hashing done by the *Trusted Platform Module (TPM)*, protect data authenticity and integrity at rest and in transit; public keys are made available to any groups that participate in verification of cryptographic results
- Private keys are safeguarded in hardware; the adversary cannot physically access the internals or influence the operations of the TPM without drawing intolerable levels of attention
- The TPM correctly implements the *Trusted Computing Group (TCG)* specifications and does not leak information, including any private keys or ownership information in part or in whole
- The computing platform BIOS, CPU, and specialized *Trustworthy Computing (TC)* modules perform as intended; system memory cannot be corrupted by internal electronic components of computing platforms; physical scanning equipment, where applicable, is aligned correctly and is otherwise in good working order
- Reasonable physical security of election technology is enforced prior to the election, heading off sophisticated physical delayering attacks against the TPM [1]
- Ballot designs are correct, ballots are printed correctly, and are free from errors that may confuse voters or contain subliminal channels
- Application and platform operating software is free of critical bugs or supply chain trap doors, and does not become compromised during runtime; software is adequately inspected and binaries are correctly generated from reviewed software and securely

[1] Although the TPM resists some physical presence attacks, it is not designed to resist sophisticated hardware attacks to keep costs low. Some crypto units such as the IBM 4758 are "tamper-responsive," containing special technology to defeat tamper attempts, but these cost thousands of dollars each compared to dozens of cents for a TPM [51].

transferred between the vendor, *Independent Testing Authority (ITA)*, and *Election Authority (EA)*

- Receipts are secure: obtaining information about a voter's preferences given the voter's receipt is computationally infeasible. Also, any scanning equipment cannot read invisible ink (where used), and any confirmation codes on receipts do not contain subliminal information that can influence or confuse voters
- The poll booth is free of cameras, covert microphones or speakers, networking equipment or anything that can allow communication or observation between an external attacker and the voter

3.4.2 Trusted Roles

The roles defined in Section 2.1.2 (page 14) are entrusted with carrying out election procedures and enforcing policies. Subversion of the *Election Authority (EA)*, *Tallying Authority (TA)*, *Precinct Judge (PJ)*, or *Independent Testing Authority (ITA)* can lead to election system compromise.

3.5 Attack Mitigation

Each of the attack types are defended in closely related ways by the designs we present. At a high level, our designs protect data from modification by digital signatures performed inside of secure hardware. Digests covering the entire election data storage area prevents unauthorized insertion or deletion of single records, and authorized platform signatures prevent slipping rogue votes, not created on an authorized and activated terminal, into the main tally.

The voter verifies platform attestation evidence to ensure that the platform booted into the correct operational state. If the right software was booted, then we know that persistent malwarecannot perform subliminal, misrepresentation and other attacks designed to confuse the voter. Verifying operational state also thwarts discreditation attacks; if the platform is running the correct software, then correct software will not disclose information that can be used by the attacker to initiate false challenges. (An interesting side-bar is how we protect against non-persistent software attacks, those that exploit buffer overflows to inject code into live running machines. This special case is discussed in Section 5.7, and requires a special software module to watch a virtualized operating environment for changes to its fundamental memory structures.)

Uninformed privacy attacks are countered by using encryption to prevent disclosure of receipt verification codes to unauthorized parties. Although we cannot prevent voting queue observation attacks, we can use randomness created by trusted hardware to reorder the recording of votes into persistent storage, preventing an insider from assigning recorded votes to observed voter order.

Informed privacy attacks (*e.g.*, coercion and vote buying) are mostly handled by the higher level voting systems that use our TC designs, but we apply a technique called *late binding* to prevent receipt codes from being shown to the voter until the time that he casts his vote. By guarding the verification secrets from the voter, he cannot take interactive instructions from an adversary and walk away with verifiable proof that he performed the adversary's bidding.

Procedural attacks are prevented by using passwords on TPM signature keys and key deletion delegations to enforce election policy. When the polls open, the EA releases a password that allows the TPM to sign vote records. Prior to the EA's release of the password, election keys cannot be used to sign votes, heading off day-before attacks. At the end of

election day, the EA releases a delegation password that destroys the private keys in the TPM, to ensure that no further votes may be signed. Destroying the key hierarchy is witnessed publicly by judges from competing political parties.

We propose the ability to defeat the hostage attack by timing the platform verification step, constraining the required channel bandwidth that the adversary can use to communicate to a captured, legitimate system. We minimize the effect of discreditation attacks by using cryptography to prove authenticity of receipts; also, by controlling the use of keys, we deny the adversary the ability to produce legitimate artifacts (*e.g.*, receipts) that contain incorrect information.

3.6 Attacks Not Countered

Procedural attacks such as behind-the-curtain attacks and chain voting are caught only by attentive poll workers. However, extensions to our work might enable just-in-time printing of ballots that would circumvent chain voting. We cannot defeat denial of service attacks, but these attacks draw too much attention to the attacker so we feel it is reasonable not to address them.

When assumptions are violated, our mitigations disappear. For instance, failure of a majority of poll workers to carry out procedures can enable all types of attack. Incorrectly designed ballots, or confusing ballots, can violate the will of the people regardless of any technological defenses.

...we may never know with complete certainty the identity of the winner of this year's Presidential election...

John Paul Stevens, U.S.
Supreme Court Justice, 2000

Chapter 4

Combining End-To-End Voting With Trustworthy Computing for Greater Trust, Privacy, Accessibility and Usability

4.1 Introduction

THE goal of modern voting research is to provide election systems that are usable, transparent, and secure. Modern *Direct Recording Electronic (DRE)* systems were designed with usability in mind, but fail to provide adequate security because they rely on software and system configurations that can be tampered with undetectably. Poised as alternatives to risky, software-centric DRE designs, *End-to-End (E2E)* cryptographic voting systems offer *software independence*—the property that the election outcome cannot be affected surreptitiously by actions of software systems. That is, the features of a software

independent system will reveal changes made by malicious or faulty software systems with high probability.

As election outcome is protected by software independence, E2E systems employ lots of software throughout their designs for efficiency and usability. For instance, all E2E voting systems rely on software to help print ballots, record the results, and forward data for tallying. Software also provides flexible user interfaces allowing the physically disabled to cast votes without assistance. Voting systems that avoid computing technology suffer diminished usability and accessibility, despite that usability is the main problem of modern voting systems. Systems that do not use software suffer usability benefits: Scantegrity, for instance, lacks a trustworthy receipt printer, requiring voters that wish to verify their votes manually record numerous codes in lengthy races. Without an electronic interface, Scantegrity currently impedes the visually impaired from verifying their votes.

Unfortunately, E2E software components suffer integrity problems in the polling place that cannot be caught until after the election is over, and privacy problems that may never be caught, countering the benefits of E2E software independence. Vote flipping by incorrect *Precinct Count Optical Scan (PCOS)* scanner software might be caught by parallel testing, but such testing cannot recreate the poll booth conditions—patterns of votes, frequency of voters—and therefore might not trigger the malicious behavior. Malicious software could strike at infrequent intervals, changing selections for voters that are not likely to check the E2E integrity results. Further, malicious software on an E2E system can easily violate voter privacy in a way that can never be detected. For example, in Scantegrity [23], malicious printer software could expose ballot codes and destroy voter privacy. Malicious scanner software could identify voters with stray marks, enabling coercion. Similarly, a compromised VoteHere touch screen could respond to a pattern of touches to disclose all votes received to that point. Malicious software can also sow confusion and undermine

public confidence in the election outcome, for example, through presentation attacks (e.g., misreading voter inputs) and discrediting attacks (e.g., planting fabricated evidence of fraud). Malicious code can also reduce system availability and reliability. Even with software independence and the preservation of election integrity, E2E systems trust software systems to enforce *privacy* and provide *efficiency*, and are therefore neither immune to privacy attacks nor quick at catching integrity attacks.

We claim that despite software independence, E2E *integrity* is not end-to-end *security,* the term *security* encompassing privacy in addition to integrity. Our research in *Trustworthy Computing (TC)* suggests that many software and misconfiguration attacks can be detected and prevented through using trustworthy hardware to verify software integrity. Since E2E systems rely on software, TC should be applied to E2E by enabling good usability and by providing defense in depth against integrity as well as privacy attacks. To realize these benefits, however, we must understand what features are enabled by TC, and where to apply TC in an E2E architecture. Our contributions include an analysis of security gaps in E2E architectures, analysis of where TC can fill some of these gaps, and proposed TC enhancements to *Scantegrity with Invisible Ink (Scantegrity)* as an example of how to realize TC benefits in a fielded E2E system.

In this chapter, Section 4.2 shows where trustworthy computing can add value to E2E systems, Section 4.3 gives focused trustworthy computing enhancements to a sample E2E system, Section 4.4 discusses benefits and related problems of the approach, and Section 4.5 concludes the ideas.

4.2 E2E Gaps in Voting System Attributes

While E2E features achieve many desirable election system goals, several gaps remain because of untrustworthy software and poor usability. Table 4.1 summarizes where TC can improve E2E in each major goal. This analysis motivates the application of TC to E2E.

Goal	**Attribute**	**TC Adds Value?**
Administration	Auditability	✔
	Ease of Administration	
	Efficiency	
	Policy Enforcement	✔
	Total Cost of Ownership	
Assurance	Accuracy	✔(w/electronic interface)
	Authenticity	✔(ballot authentication)
	Availability	✔
	Integrity	None—E2E core feature
	Privacy	✔
	Public Confidence in Dispute Resolution	✔
	Repudiated Choice, Non-Repudiated Cast	None—E2E core feature
	Small Trusted Custody Chain	✔
Voter Utility	Accessibility	✔(w/electronic interface)
	System Understandability	
	Voter Verifiability	None—E2E core feature
	Voting Usability	✔(w/electronic interface)
	Identify Problems in Precinct	✔

Table 4.1: *Trustworthy Computing (TC)* can enhance *End-to-End (E2E)* systems in varying degrees.

TC cannot benefit E2E in every attribute. Attributes marked with "E2E integral feature" are handled well by E2E, because E2E systems are specifically designed to provide these features. Unmarked attributes generally are not areas in which TC can help; specifically, TC does not benefit ease of administration (administrator training is still required), effi-

ciency (E2E systems can be efficient without trustworthy techniques), cost of ownership (TC can add extra cost), availability (computers can crash regardless of malice); and system understandability—some say that *Trusted Platform Modules (TPMs)* actually reduce transparency.

TC can benefit three critical areas. Privacy is the main area in which TC techniques add value to E2E systems: platform attestation through TPMs [97] can ensure that cryptographic operations can be carried out only when the system has booted the correct software (dynamic attestation can check certain runtime memory properties ([60]), mitigating the risk of malicious or unauthorized software disclosing the ballot codes, scanned images, or user data to attackers. Similarly, reliance on the chain of custody can be reduced by binding keys and software to a TPM to ensure that only the correct platform can access data, and only the correct data is supplied to or retrieved from the platform. Verifying correct software operation is crucial to detecting problems early—for example, a trustworthy receipt printer can reveal in the polling place that scanner software has recorded an incorrect selection, allowing the voter to discard his ballot and vote again. Voting system software can be created using formal specifications and translated directly into code, useful for proving cryptographic protocol properties [28, 47, 48].

Safer DRE designs, in turn, provide good usability which can prevent errors in the polling place [46]. In addition to catching undervotes and overvotes prior to casting, studies have shown that electronic systems are generally easier and arguably more preferred for people to use for ballot marking than directly marking paper ballots, particularly in long races with many choices [31]. For instance, mistakenly marking a single choice on a long race by directly marking a paper ballot would require *all* the selections to be re-entered on a fresh ballot, introducing transcription errors—a DRE would permit easy remarking as the choices are specified before the ballot is printed.

Other E2E systems that use DRE can be made safer through *sealed, non-migratable keys* and platform attestation:[1] if the DRE software in Benaloh's voter-initiated auditing [12] were compromised, a coercion mode could be activated by a special sequences of touches applied to the user interface to recognize the voter and bind his identity to his vote. Managing the device signature key in hardware and sealing it to the correct platform state would allow the ballot to be signed only when the correct software was running. Additionally, sealing to the TPM prevents theft of the signature key. Accessibility to disabled voters and non-native speakers is a compelling reason for DRE: computerized display and entry apparatus can accommodate differently abled voters through high-contrast displays, audible ballots, sip-and-puff systems for paralyzed voters, and multi-language ballots for non-native speakers.

Additional benefits include better enforcement of policy and procedures through TPMs; for instance, the *Platform Vote Ballot (PVB)* binding key protocol [35] signs voter choice and ballot identification data only after an administrator activates the signing key with a password revealed on election day. Better auditability is achieved by using trustworthy cryptographic logging systems in the polling booth, augmented with write-once memory.

Our position is that E2E provides good auditability, voter- and universal-verifiability, but existing E2E systems do not protect against privacy attacks carried out by malicious software. Trustworthy techniques perfectly complement E2E by preventing malicious software operations, protecting both privacy and transport integrity thereby enabling computers to safely provide accessibility to the disabled.

[1] The TPM can use a *sealed* key only when platform software measurements recorded during system boot match specified values. A *non-migratable* key can be used only by the TPM that created it.

4.3 Enhancing Scantegrity

To illustrate some of the ways TC can enhance E2E voting technology, we propose some modifications to the Scantegrity voting system.[2] Although we focus on Scantegrity, privacy and many other issues apply evenly to all E2E systems—we chose Scantegrity as our example because we feel it is the easiest to understand and the most likely to be adopted. In fact, Scantegrity has been used in the Takoma Park, Maryland mayoral race of 2009 [16].

4.3.1 Software Components of Scantegrity

The software components of Scantegrity are described in Figure 4.1. The privacy risks occur in the trusted workstation, which creates and manages the commitments, and in the ballot printer. Both systems are entrusted to protect the mapping of commitment codes to ballot serial numbers.[3] Additionally, the scanner tabulator can recognize stray marks or identifications and reveal full ballot images to attackers. Malicious software can be surreptitiously installed to carry out these actions.

[2] Independently, the Scantegrity team is investigating how to apply trustworthy computing to the system.

[3] An attacker with the mapping could read the ballot serial number from the voter's receipt and look up the ballot codes on the public server to reveal the voter's selections.

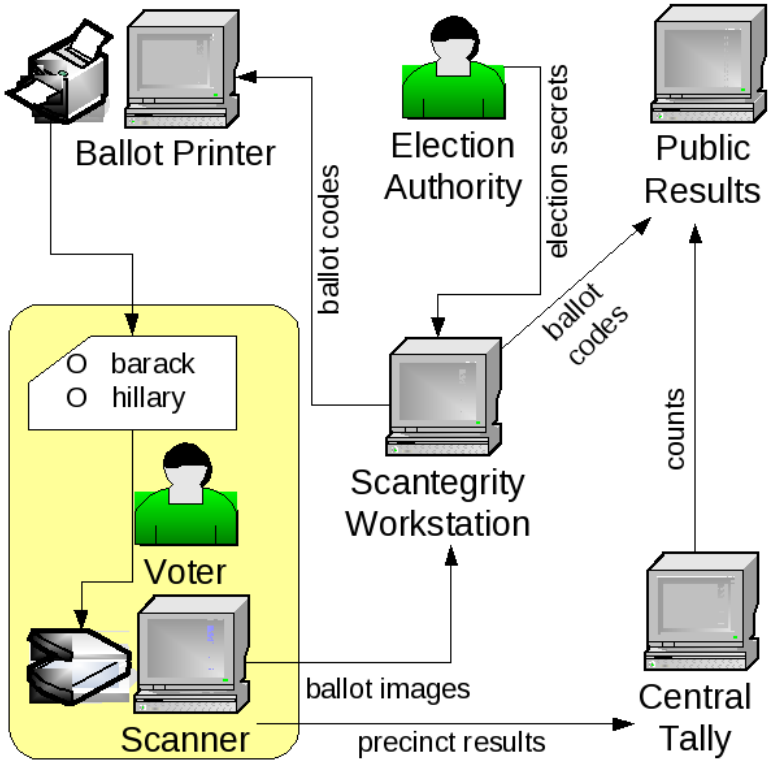


Figure 4.1: The software components of Scantegrity are represented as PC workstation icons, and include Ballot Printer, Precinct Scanner/Tabulator, Scantegrity Workstation, Central Tally Server, and Public Results Systems. The Scantegrity Workstation produces commitment codes bound to ballot serial numbers to supply to the ballot printer. The ballot printer prints the commitment codes on the ballot, and the voter reveals the codes when marking his ballot. The scanned ballot images are sent to the verification server, where the voter can confirm that the system recorded his marked ballot correctly by checking his codes, and where anyone can verify the tally was computed correctly.

4.3.2 Incorporating Trustworthy Computing

To illustrate some of the ways TC can enhance E2E voting technology, we propose some modifications to the Scantegrity voting system (our focus is on the types of changes that can be made; full technical details are explored in other chapters):

1. Introduce trustworthy DRE marking units—replace the scanner and ballot printer with a TPM-equipped DRE running the PVB protocol [35]. The DRE will solicit the voter's selections, decrypt and print ballot codes, and record and sign votes and vote storage with sealed keys. A signature made by a sealed key is evidence that the platform booted the correct software. Using DRE improves usability and reduces errors by ensuring that everyone marks ballots in a consistent way.

2. Use TPMs for secure software provisioning—allow the software to be installed only on authorized commitment, ballot printing, and ballot scanning systems by using non-migratable, sealed keys. The independent testing authority compiles and encrypts certified software for units with specific TPMs. Protecting software with non-migratable keys prevents tampering in the software supply chain or software theft by the adversary.

3. Encrypt commitment codes with TPMs and non-migratable sealed keys—protect the commitment database codes with a key sealed to a specific ballot printer and verification server.[4]

4. Add cryptographic signatures to the ballot printer—using a sealed TPM key, the ballot printer could print a signature of the ballot codes with a key sealed to its TPM to prove authenticity. Sample ballots could be intentionally spoiled to verify their codes against the printer's public key.

5. Add a trustworthy receipt printer to the Scanner Tabulator—use a PVB-like binding protocol to reveal and print lettered codes to voters. The tabulator can sign the cast vote with a sealed signature key providing evidence that the correct software was

[4]The printer could record encrypted commitment codes on the ballot with *Two-Dimensional QR barcodes (qrcodes)*, simplifying the data provisioning process.

running and that the optical marks were sensed correctly. If errors are detected, the voter discards his ballot and votes again. A receipt printer also improves usability and speeds up the election process by printing the code numbers for voters.

4.4 Benefits and Problems

The main benefit of adding TC to E2E is guaranteeing privacy by binding public-key cryptography to correct software state to protect the commitment codes and scanned ballot images from exposure to malicious software. The receipt printer catches problems early in the polling place, rather than after votes are tallied, providing usability in a secure way. An all-DRE solution could provide good usability while enforcing procedures to prevent day before attacks as explained in [35]. Integrity and authenticity can reduce the size of the trusted chain and remediate lifecycle attacks, benefiting system deployment, ballot printing, precinct data collection and transport. TC can help mitigate software attacks, providing effective systems security.

The main cost of TC is more complicated engineering design and key management. The use of TPMs assumes a trusted third party key distribution mechanism that may cause a deployment problem for the voting community. Further, sealed non-migratable keys must be created directly on the platforms in question, involving direct interactions to take *ownership* of the TPM on possibly thousands of units. These interactions could be made more practical using an assembly-line process in a secure facility, as ownership of TPMs must be done only when the systems are first procured—fresh keys can be regenerated at each election, but trusted software could do this inside a secure facility.

Certain aspects of TC are new, and certain techniques such as formal methods are hard to understand. For instance, proof checkers can readily discover flaws in formally specified

software, but designers may struggle with the formal specification syntax, some proof checker tools are hard to use, and the independent testing authority might find the tool's results hard to understand.

4.5 Conclusions

All voting systems used in large scale elections rely on software for efficiency, usability, and accessibility, but software carries risk (especially for privacy) even for software independent verification systems such as E2E. From the main voting goals—ease of administration, information assurance, and usability—E2E cannot fully satisfy certain aspects without the help of trustworthy computing techniques. In particular, trustworthy computing increases privacy and accessibility, and helps voters catch problems in the polling location, making voting safer and better for everyone. In conclusion, we must work hard to improve the state of software used in election systems, and we can produce a much better system by incorporating trusted computing into E2E designs.

Part of the inhumanity of the computer is that, once it is competently programmed and working smoothly, it is completely honest.

Isaac Asimov

Chapter 5

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

5.1 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 [46]. Unfortunately, bad security engineering of existing products (*e.g.*, [11, 14, 15, 32]) 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 chapter describes in considerable detail how to design a more trustworthy DRE for achieving outcome 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 on-board nonvolatile memory and counters—to create and manage a special signature key called the *Platform Vote Ballot (PVB)* signature key. The PVB binds together the booted state of the platform, the ballot presented to the voter, and the voter's cast vote, and thwarts unauthorized modification, insertion, or deletion of votes.

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, administrators deliver the signed storage to tallying officials who 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. This chapter presents our protocol with enough detail to demonstrate feasibility of an actual implementation on a system compliant with the *Trusted Software Stack (TSS)* published by the *Trusted Computing Group (TCG)* [97].

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, allowing election policy to be enforced by hardware and preserving data integrity and voter privacy. We designed it with these features:

- 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
- Cryptographic binding of vote to ballot—signed proof that a specific ballot guided the voter's decisions is available for verification
- 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])
- Cryptographic integrity and privacy—integrity is enforced by public key cryptography, and privacy is preserved by 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 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 that the 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 his 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,

[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 [29].

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 multi-language 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 [61]. 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, butterfly markings, 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 as stated in Section 5.3.2, 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 chapter is organized as follows: Section 5.2 briefly reviews previous and related work; Section 2.3.2 introduces the capabilities of a TPM; Section 5.3 describes a notional system architecture and states several security assumptions; Section 5.4 presents our protocol;

Section 5.5 analyzes the protocol, including its security and special features; Section 5.6 discusses benefits and limitations; Section 5.7 presents future work, and Section 5.8 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 protocol is an example of building secure systems with trust rooted in TPMs.

5.2 Previous and Related Work

Arbaugh [9] suggested using TPMs in voting by outlining an on-line protocol for attesting systems through a central server. Rössler, *et al.* [73] 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 [67] sketched a voting system architecture incorporating TPMs, but the TPM's role assures only presence of correct software—the platform state is not bound to the cast ballot.

Yee [101] 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 [54], suggests using a hardware security module to protect chained digital signatures but not signature keys, and propose light-weight voting software booted from a CD-ROM to eliminate reliance on pre-installed software and hardware. The security of any system that obtains software and private keys from removable media is vulnerable to compromise through theft 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.* [32] 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 [97]. Pearson *et al.* give a slightly dated but comprehensive overview of TPMs and the TCG [68], and Challener [19] provides an excellent practical guide to the TPM for software developers. Additionally, TrouSerS [50] is an open source implementation of the TSS and includes test suite software useful for understanding programming interface, while Strasser [95] provides an open source TPM emulator to aid development.

Previous authors have applied TPMs to non-voting domains. Sevinç [86] 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 [55] catalog of DRE attack strategies, a threat analysis derived from attack trees by the Brennan Center [66], and research analyses by Kohno *et al.* [58], SAIC [82], RABA [71], and Compuware Corporation [27] on flawed commercial DRE implementations. Additionally, Hursti [49] analyzed the problems of unauthenticated software installs, and

Feldman, *et al.* [32] analyzed the damage caused by viruses when policies and procedures are not followed. Additional vulnerabilities in modern DREs were uncovered in the EVEREST Project [15] and in the California Top-To-Bottom Review [14].

Unfortunately, most critiques on 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 (VVPAT)* (*e.g.*, [42]). 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 (E2E) systems (*e.g.*, [24, 25]) provide strong assurance to the voter that his 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 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 [34].

Some researchers have raised questions of how trustworthy the TPM is. The 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 non-compliant, only one of those implemented the current version (1.2) of the TPM specification [76]. Although correct TPM implementation is critical to a protocol like ours, there are many different TPM vendors to choose from, and the specific

problems identified in the single 1.2-compliant TPM discovered by Sadeghi 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 [44]; data can be modified on disk without added protections such as full disk encryption [83]; and rogue software can be loaded in a way that is undetectable by antivirus products [74] 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.

5.3 Architecture

We now present our design. 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.

5.3.1 System Elements

Figure 5.1 shows the system elements in the context of the high level architecture.

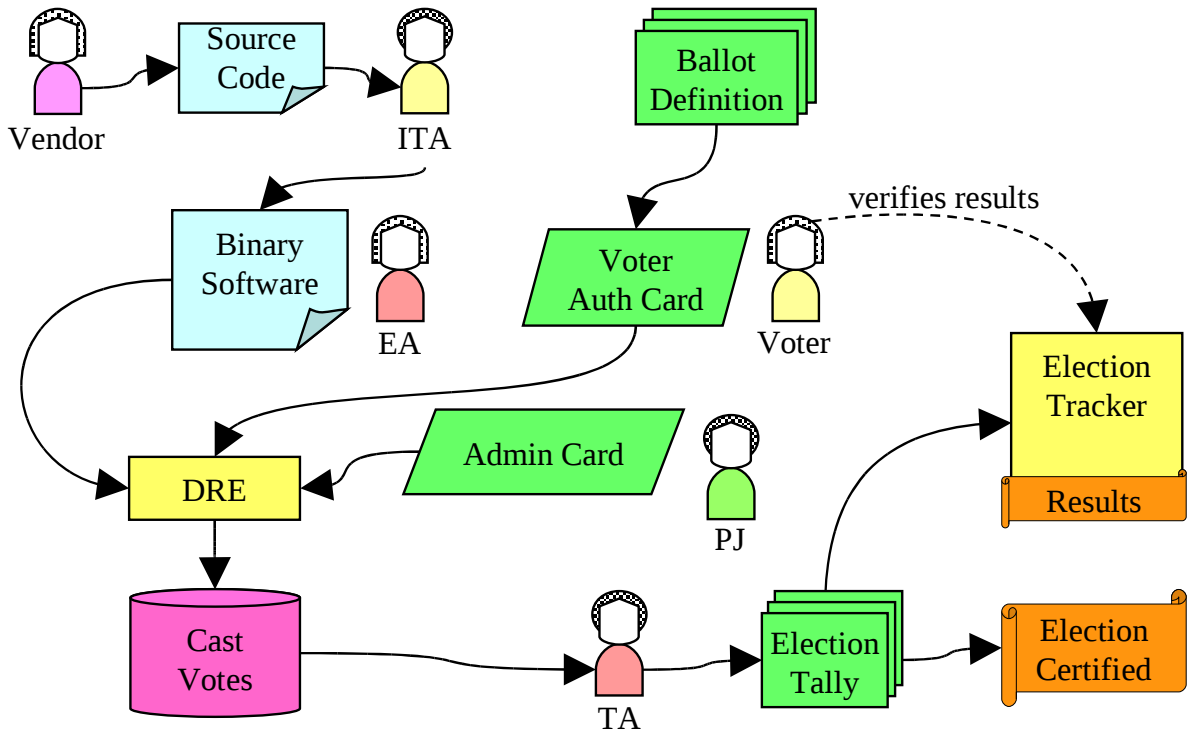


Figure 5.1: The architecture includes: DRE terminal (with a TPM), tallying systems, and tracker/reporting systems such as Election Tracker [89]; trusted authorities, Election Authority (EA), Precinct Judge (PJ), Tallying Authority (TA), Independent Testing Authority (ITA), the voter; and binary images of software, ballot, and storage for cast votes.

5.3.2 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:

- Asymmetric digital signature keys and hashing afforded by the TPM adequately ensure data authenticity and integrity during storage and transmittal

- The chain of trust of *Platform Configuration Register (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 *Basic Input/Output System (BIOS)*
- The TPM and other trusted hardware are operating correctly
- The software components that form the measured chain of trust are behaving as expected (correctly implemented, correctly executing)
- A pseudorandom index is sufficient to protect voter identities against analysis of stored vote order
- 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
- 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, *Independent Testing Authority (ITA)*, and *Election Authority (EA)*; and that the ballots are reviewed for accuracy.

5.3.3 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 parts, but a short summary follows:

- *Election Authority (EA)*—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 EA is critical to the entire protocol

- *Tallying Authority (TA)*—receives encrypted, completed ballots, tallies them, and produces the general results

- *Precinct Judge (PJ)*—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 PJ 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 PJ can conduct. (Denial of service is still possible by a rogue or poorly trained PJ.)

- *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.

5.4 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 parts 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. Figure 5.2 highlights the main features of the protocol.

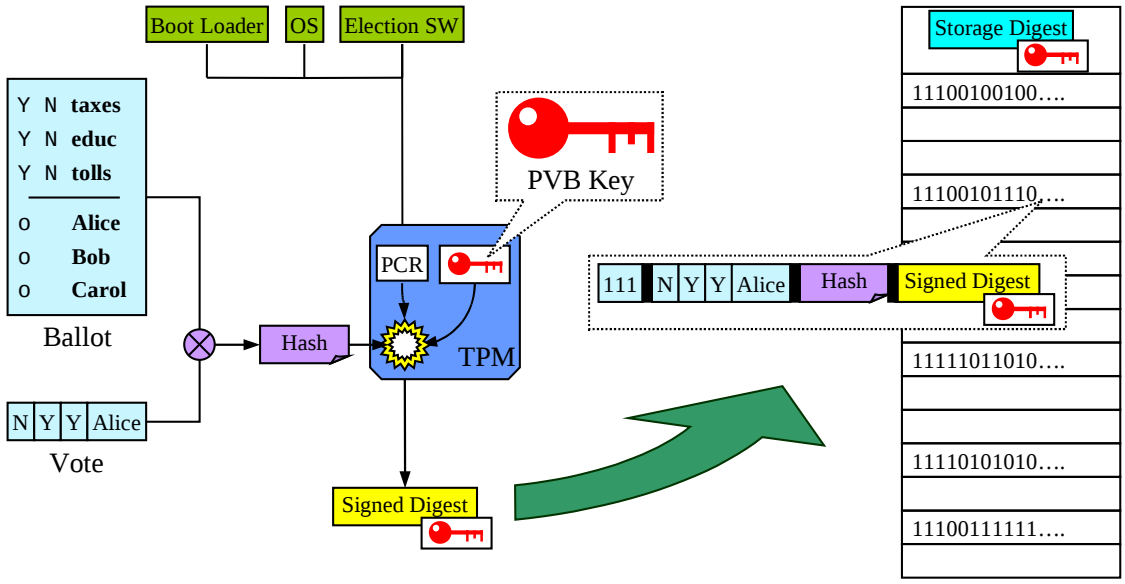


Figure 5.2: The protocol loads the PVB key into the TPM to produce a signed digest of the vote and ballot. The PVB is usable only when the PCRs match a specified set of measurement values. The vote, digest, and hash are stored in a pseudorandom location on disk and the storage is signed whenever a new vote is inserted.

5.4.1 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 numerous

arguments, and others require additional setup/take down 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 [97] and the associated TSPI implementation. Certain protocol features such as sealing are not available on prior versions of the TPM.

5.4.2 Protocol Actors

Several actors and software items referenced in Section 5.4.3 include:

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

5.4.3 Protocol Parts

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

Part 1 - Platform Initialization, Software Load, and Key Creation

$$EA \rightarrow Platform : \textbf{TakeOwnership}(ownerPass, srkPass), \quad (1.1)$$
$$Platform \rightarrow Storage : meta_{SRK}$$
$$EA \rightarrow Platform : \textbf{CreateDelegation}(SRK, \quad (1.2)$$
$$\text{DELEGATE\_LoadKey}, srkPass, pollopenPass),$$
$$EA \rightarrow Platform : \textbf{CreateDelegation}(Owner,$$
$$\text{DELEGATE\_OwnerClear}, ownerPass, pollclosePass)$$
$$EA \rightarrow Platform : \textbf{Key\_CreateKey}(PVB, srkPass, \quad (1.3)$$
$$pcrComposite),$$
$$Platform \rightarrow Storage : P_{SRK}(PVB), meta_{PVB}$$
$$Platform \rightarrow Storage : S_{PVB}(h(VoteStorage)) \quad (1.4)$$
$$Platform \rightarrow TA(\text{via } EA) : P_{PVB} \quad (1.5)$$
$$TA \rightarrow Storage : P_{TA} \quad (1.6)$$

1. The TPM is physically reset to erase any previous ownership. SWinit is installed and booted. The EA chooses a password for the platform's TPM, *ownerPass,* and a password for the *Storage Root Key (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 EA creates two delegations: a load key delegation, protected by *pollopenPass*, allowing the PJ to use the SRK to load the PVB; and an ownership delegation, protected by *pollclosePass*, allowing the PJ only to clear ownership in the Voting Termination part. These passwords are kept secret until election day. Delegation grants only the needed rights to the PJ without disclosing the full-use passwords, maintaining least privilege.

3. The EA 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.[2] 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 EA signs the empty vote storage area with the PVB.

5. The TPM exports the public portion of the PVB to the EA, and the EA securely transmits the public key to the tallying authority. (PKI could create an integrity-protected channel between the EA and TA in this step, to ensure that the TA receives the correct public key.)

6. The TA's public key is installed on the platform to encrypt the storage during Voting Termination—Precinct.

   Finally, the EA 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.

[2]The audit log is not central to the use of the TPM, but is necessary for verification. All actions of the protocol should be logged. Section 5.7 discusses important issues related to secure auditing and logging.

Part 2 - Election Day Initiation (and Reboot)

$$EA \rightarrow PJ : pollopenPass \tag{2.1}$$
$$PJ \rightarrow Platform : \textbf{LoadKey}(PVB, pollopenPass) \tag{2.2}$$
$$\text{Verify } h(VoteStorage) = P_{PVB}^{-1} h(VoteStorage) \tag{2.3}$$

1. The EA reveals the delegation *pollopenPass* to the PJ. This delegation password allows the PJ only key loading, vice unrestricted access to the SRK, and can be posted publicly.

2. The PJ 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: (1) *pollopenPass* is entered, and (2) the PCRs match the measured PCR values of the certified SWvote software. If either condition fails, the PVB key cannot be used and the PJ 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).

Part 3 - Voting and Recording

$$Voter \rightarrow Platform : vote \tag{3.1}$$
$$i \leftarrow \textbf{RANDOM}(1, \text{sizeof}(VoteStorage)) \tag{3.2}$$
$$Platform \rightarrow VoteStorage[i] : vote, S_{PVB}(h(vote \parallel ballot)), \tag{3.3}$$
$$Platform \rightarrow Storage : S_{PVB}(h(VoteStorage))$$

1. The voter receives an electronic ballot from the PJ (possibly via voter registration card) and presents it to the platform. The platform displays the ballot to the voter, and the voter commits his 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/or PJ 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 PJ keeps counts of how many people attempted to vote and completed voting on each DRE.

Part 4 - Voting Termination—Precinct

$$EA \rightarrow PJ(\text{and } TA) : pollclosePass \tag{4.1}$$

$$Platform \rightarrow PJ : P_{TA}(VoteStorage, \\ S_{PVB}(h(VoteStorage \parallel pollclosePass)), P_{PVB}) \tag{4.2}$$

$$PJ \rightarrow Platform : \textbf{OwnerClear}(pollclosePass) \tag{4.3}$$

1. The EA reveals *pollclosePass* to the PJ and the TA (used later). The PJ 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 TA's public key. (Omitting, or submitting an invalid *pollclosePass* in this step reveals premature precinct termination to the TA.)

3. The PJ 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 (TA).

Part 5 - Tallying

$$\text{Decrypt} : P_{TA}^{-1}(VoteStorage, \quad (5.1)$$
$$S_{PVB}(h(VoteStorage \parallel pollclosePass)), P_{PVB})$$
$$\text{Verify } h(VoteStorage \parallel pollclosePass) \quad (5.2)$$
$$= P_{PVB}^{-1}(h(VoteStorage \parallel pollclosePass)$$
$$\forall i \in \{1, 2, 3, \ldots\}, \text{Verify } h(VoteStorage[i]) \quad (5.3)$$
$$= P_{PVB}^{-1}(h(VoteStorage[i]))$$

1. The TA decrypts the transported data. The TA looks up the supplied public PVB key against that supplied earlier by the EA—if the key is not known, halt and report the error.

2. The TA 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 TA verifies each recorded vote in the VoteStorage area—failure indicates a corrupt vote.

   The TA 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 TA 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.

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.

5.4.4 Protocol and Implementation Enhancements

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

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 TA 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.)

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 TA, 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.

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 [93]. 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 TA public key.

5.5 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. He 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 post-election precinct data bundled for transmission to the tallying location.

We assume that the adversary does not know the PVB private key, and is not able to recover the private key. We assume that he cannot physically access the internals or influence the operations of the TPM without drawing significant attention.

5.5.1 Countered Attacks

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

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 TA 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 his intent, but this attack violates our assumption about certain trustworthy hardware.

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 TA would detect the difference on the vote and storage area digests when verifying the PVB signature (as recorded by the TPM), revealing the attack.

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

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. The case of (2) cannot be mitigated without write-once storage. An adversary that subverts a running platform after the PVB is loaded can replace the vote storage and command 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 5.7.

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 does not match the order of voter interactions (with high probability).

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 certain threshold is achieved.

5.5.2 Attacks Not Countered

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

Hardware Attacks: These are specific hardware attacks that affect any complex software and hardware system:

- 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 [58].

- Device tampering—misrepresenting a ballot to a voter can cause him to cast a vote that opposes his intention. Our protocol cryptographically binds the contents of the ballot with the vote, but there is no way to prove what the voter actually saw when making his decision.

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 PJ 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.

5.6 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 collection of the vote, 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:

- Readily implemented with the TSPI through prototype code.
- Works for any TPM implementation and platform with TPM support
- 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 capabilities such as *Intel Trusted eXecution Technology (TXT)* extensions

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 [60]. Software independent E2E systems also offer protection against software faults.

Last, the design in this chapter provides no assurance to the voter that the machine is an authorized device or is configured or behaving correctly. This specific issue is treated in Chapter 6.

5.7 Future Work

This protocol 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.* [80] refers to this as *clobs*—count-limited objects. Another task would be to prove the protocol properties formally 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 [24, 25] empower each voter to verify that his vote was correctly

recorded and tabulated, 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 cryptographically verify historic system state and event occurrence, preventing surreptitious wholesale replacement of the voting storage. The challenge is determining what data to record to balance the competing requirements 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 [60]. Advances in both of these areas would benefit both electronic voting and information assurance in general.

5.8 Conclusions

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.

I can't explain myself, I'm afraid, Sir, because I'm not myself you see.

Alice, from Alice in Wonderland
(Lewis Carroll)

Chapter 6

A Human Attestation Protocol for Trustworthy Electronic Voting: Bootstrapping Trust Using TPMs, Smart Cards, Timings, and Scratch-Off Codes

6.1 Introduction

In the voting and computing industries, managers and policy makers desire the benefits of electronic technology but are hesitant to accept the security and privacy risks associated with using complicated computing platforms that may be vulnerable to compromise, preferring instead to use simpler systems that are perceived to be more trustworthy. Electronic voting, like other general purpose computing applications, must ensure user privacy and vote integrity in the face of malicious software taking control of a user's platform or even a determined adversary replacing the user's platform with one under the adversary's control.

One feature that is consistently lacking in modern computing technology is the ability for a *human* to verify the correctness of the software state of the platform prior to entering sensitive information into it.

We provide a protocol through which the voter can verify the software state of an electronic voting machine. To do so, the voter uses a trusted smart card and scratch-off sheet of result codes, both provided by what we call the *Challenge Authority (Challenge Authority (CHA))*. The voter may also verify the smart card through a timing test, and he may audit the scratch-off sheet. Thus, our method bootstraps trust from the CHA, to the smart card, to the voting machine.

In the protocol, a *Trusted Platform Module (TPM)* in the voting machine stores measurements of the configuration and software that the voting machine booted. The smart card verifies the measurements cryptographically. Whereas the TPM design of Fink *et al.* [35], used to secure *Direct Recording Electronics (DREs)* voting systems, produces an official record that the election authority can verify was generated by a valid voting machine after the election, our protocol enables the voter to confirm the validity of the voting machine before entering any sensitive data (*e.g.*, making selections, casting his vote).

There are advantages to using an ordinary and inexpensive smart card for bootstrapping trust versus having the voter interact directly with the voting machine. First, the card can perform calculations helpful to verifying certificates issued by the voting machine's TPM. Second, the smart card can perform sensitive timing measurements helpful in detecting "proxy attacks" (*e.g.*, in which a corrupt election authority presents an evil voting machine that communicates with a hidden valid machine).

Our approach overcomes two significant challenges: communicating with an ordinary smart card (one that lacks any sort of built-in display, *e.g.*, an LCD display or indicator light), and verifying the smart card. First, to communicate its results to the voter, the smart card

sends a one-time "results code" to the potentially hostile voting machine for it to display to the voter. Using the scratch-off sheet, the voter interprets the displayed code.[1] Because the voting machine does not know what the code means, and because it cannot fabricate valid codes, it cannot interfere with this communication without detection.

Second, after being issued a smart card, the voter may choose to verify the software running on it. To do so, the voter can perform a timing verification step using a trusted laptop provided by the challenge authority. Because the smart card has limited capabilities, such timing techniques (including *e.g.*, Pioneer [84] or Endor [39]) can be used without incurring the major drawbacks that arise when applied to full voting machines.

Our contribution is a protocol with these innovative features:

- Smart cards used for attestation do not require built-in displays or keypads, safely using the display and input devices of the untrusted platform
- The voter verifies the platform authenticity and software state prior to the voter entering any private information
- The protocol detects a specific type of proxy attack that we define in Section 6.5.1 through timing measurements of the attestation process

Our attestation protocol can be combined with other voting techniques, including *End-to-End (E2E)* technologies and ballot-marking machines for paper ballots. Whereas E2E systems (*e.g.*, Scantegrity [23]) provide strong election outcome assurance, they detect potential problems late in the process and do not guarantee voter privacy (*e.g.*, a corrupt scanner could violate ballot privacy).

Our protocol is designed for electronic voting but can be easily adapted to general-purpose mobile computing applications.

[1] The scratch-off sheet keeps the attestation decision string a secret until needed, preventing unauthorized disclosure to adversaries in advance of the attestation step.

This chapter is organized as follows. Section 6.2 reviews prior and related work; Section 6.3 discusses the adversary threat model; Section 6.4 provides the protocol in detail; Section 6.5 presents the security claims and analysis, including timing constraints that must be present to detect the proxy attack; Section 6.6 describes an alternative to smart cards; Sections 6.7 and 6.8 discuss future work and concludes the ideas.

6.2 Previous and Related Work

The field of trustworthy computing is proliferating due to information protection requirements and digital rights management. One of the products of groups such as the *Trusted Computing Group (TCG)* is the TPM, an embedded cryptographic processor and non-volatile storage device that can generate keys and use them securely. Additionally, TPMs support advanced features such as key migration, enabling one authority to provide a platform and a different authority to create a key to be used on it. Our protocol uses these features to ensure that the platform is the correct one and that its software is in the correct state. The TPM is explained in specifications [97], and the software programming interface is explained in [98]. Although reading the TPM specifications is challenging, there are several resources available to help navigate the functionality [19, 75] and the authors find that browsing the code in the test suite [50] is useful when programming the API.[2]

We time the sequence of challenge/response operations—a process we call a *timing side-channel*—to verify a resource-constrained smart card that later attests the TPM, achieving bootstrapping of trust. Seshadri *et al.* created Pioneer [84] to verify the software integrity of general purpose computing platforms by timing how long a self-verifying checksum routine

[2]Further, IBM recently released their 1.2-compliant TPM emulator [52] that comes with a command line environment which can be used to prototype TPM protocols, or verify how to do things with the TPM. A different emulator is in [96].

takes to answer a challenge. Pioneer is, however, plagued by too many assumptions: no communications with a third-party host; a single CPU (vice multicore) architecture; the challenge transmission is instantaneous;[3] the verification code is optimal. Gardner *et al.* proposed Endor [39] that exploits main memory latency to expose substitution attacks in fewer processing steps than Pioneer, but it, too, suffers from impractical assumptions: no multithreading, single core CPU, no proxy attack, and no *System Management Mode (SMM)* interrupts. Seshadri *et al.* [85] propose the SWATT timing approach but stipulates the use of a resource constrained environment such as embedded systems microcontrollers. It is their suggestion, and the strict assumptions made by the other approaches, that led us to consider time-based attestation of smart-cards as a first step in verifying the software running on a general purpose computing platform. In further support of timing side-channels for attestation, Franklin and Tschantz [38] prove that tamper-evident data and control programs are possible as long as a synchronous channel—one with a known minimum delay—is used between a verifier and a prover.

One feature that is consistently lacking in modern computing technology is the ability for a human to verify the correctness of the software state of the platform prior to entering sensitive information into it. *Trusted computing* supports attestation by using TPM to engage in a special challenge/response protocol with networked computers, called *Trusted Network Connect (TNC)*. Although useful for the data center, TNC runs only when the platform attempts to connect to the network, and therefore does not support the mobile, "hotel room" scenario where the TNC back-end is not available. In our approach, the user can verify a non-networked platform using a smart card previously verified with a simple stopwatch.

[3] to avoid the possibility of precomputing the responses before the user enters the final character of the challenge

TPMs have been proposed for voting. Paul and Tanenbaum [67] sketched details of how a TPM can provide attestation evidence of software for voting, while Fink, Sherman, and Carback [35] designed the *Platform Vote Ballot (PVB)* binding protocol in which the TPM signs cast ballots with a PVB key that loads only when the correct software is booted on the platform. The PVB design ensures integrity of the cast ballot and correctness of the voting platform software state, but it provides no such assurance to the voter. While no protocol cannot detect subtle hardware errors in the voting booth, our design can rule out the malicious terminal and proxy terminal scenarios by proving the integrity and authenticity of the platform to the voter immediately prior to him entering his vote. Our protocol can fit in nicely with the PVB DRE system, or other DRE-based systems like VoteBox [78] and it can work with non-voting systems; for instance, it might be a nice complement to the *National Security Agency (NSA)*'s High Assurance Platform program [62] and various vendor implementations, *e.g.*, [40] where tactical warfighters want proof of the platform state before interacting with mobile, non-networked systems.

Our protocol requires the voter to compare response characters (also called *strings*) to verify the attestation decision made by the smart card. Some voting systems such as [3] have assumed that the voter is capable of doing string comparisons, adding minimal usability burden to the voter. We require the use of scratch-off cards to keep the response strings secret until needed during attestation. Since other voting systems such as Adida's Scratch & Vote [5] use scratch cards, we incur no additional usability issues compared to these systems. Kelsey describes attacks against scratch card systems used for ballot commitments in [56], but our protocol does not tie any ballot commitment to the revealed secret; therefore, knowledge of the secret is useless in a vote-buying attack.

Mutual authentication using smart-cards was examined by Abadi *et al.* [2], in which a *Personal Identification Number (PIN)* and proof of a private key authentication the user

to a server and the server used a certificate and a private key to authenticate to the user. However, the server private key was not bound to the software state of the platform, enabling a malicious computer program to take over the server and compromise the user's privacy. The TPM quote feature used in our protocol provides measurements and cryptographic proof of the software state of the platform, mitigating this risk.

Much has been written on the problem of software injection attacks against electronic voting systems and on DRE in particular (*e.g.*, [11, 14, 15, 32]). Some interesting work that mitigates software compromise include Yee's [101] reducing the size of the software stack and Jorba's *et al.* [54] Scytl architecture that uses hardware security modules to protect chained digital signatures. Others prefer not to trust the voting terminal and advocate E2E voter verifiable voting systems [23, 25] that prove to a voter his vote was cast as intended, recorded as cast, and counted as recorded. Fink and Sherman [34] note that end-to-end integrity is not end-to-end security, and argue that combining TPM methods such as PVB in parts of the election process improve privacy thereby improving overall security. Our human attestation protocol builds on this idea by ensuring privacy through software attestation prior to the voter indicating his preferences.

6.3 Threat Model

We define the adversary's motives and technical ability, the voting environment, and specific threats.

A *correctly configured platform* runs authorized software and uses authorized data files. It boots the authorized operating system software, starts the correct voting software, interacts with authorized devices including smart cards that typically help initiate the voting session, presents the authorized ballot to the voter, and records the vote into storage without any

changes, deletions, or duplications of the vote or previous votes. Correct configurations affect only the software and configuration files; a voting platform must ensure *trusted paths* between the hardware components, including the display, system and I/O bus, memory, CPU, and the user input device.[4]

We define the adversary as part of our attack model. The adversary wants to compromise voter privacy and disrupt the intent capturing process that records the voter's choices. The adversary is technically proficient, has access to legitimate voting terminals before and during the election, and has access to the software source and compiled binaries. The adversary has physical access to an actual voting terminal before, during and after an election, and owns a replica voting terminal that can pass as a real one. The adversary is risk-averse, wanting to avoid detection where possible while meeting as many goals as possible.

The adversary is assumed capable of, and willing to execute the following attacks:

- corrupting the voting platform software or ballot definition files
- corrupting or substituting a smart card used to initiate the voting session
- deploying an unauthorized machine
- deploying an authorized machine under adversary control
- deploying an unauthorized machine that can communicate to an authorized machine under adversary control

We assume the polling location consists of a private polling booth in which the voter interacts with the terminal and casts his vote in privacy. The poll booth consists of a single, stand-alone voting terminal connected to an electrical power source. Entry to the booth is

[4]Failure of a *trusted* system component to function as intended may cause the security controls to fail.

monitored by election officials, and entry is restricted to the voter or a group of two or more administrators (with competing interests) during the open poll hours.

We assume that the adversary wants to avoid detection and excessive cost, making certain attacks undesirable. One such attack is a "hidden camera" privacy attack that requires a breach of the voting booth integrity, adding extra physical hardware that could be detected. The denial of service attack, caused by destroying voting equipment or the polling location, is trivial and is too risky and detectable for our adversary, in addition to being too obvious. We assume that a majority of the poll workers are "good," following correct procedures and halting the election upon detecting any malicious activity.

Additionally, the adversary cannot corrupt all makes and models of smart cards, nor force a majority of smart card makers to assist him with his attacks.

Ours and other protocols make use of trusted cryptographic hardware such as the TPM; therefore, we assume that the adversary cannot learn any private keys, in whole or in part, created within the TPM. This assumption is made based on the difficulty of factoring the *Rivest Shamir Adleman (RSA)* moduli and the extreme technical problems of physically de-layering a TPM and probing it during operations to recover the keys.

6.4 Attestation Protocol

We now describe the attestation protocol in terms of assumptions (components, actors), assumptions, and specific steps.

6.4.1 System Overview

Architecture

The architecture consists of: a general purpose computer (PC) voting platform that has a TPM; a smart card; and scratch off cards that hold verification decision secrets. The smart card is able to load data and attestation software, process TPM quotes of *Platform Configuration Register (PCR)* values, perform timing measurements of attestation exchanges, generate random numbers, and communicate verification decisions to the PC. The PC accepts voter input, exchanges data with the card, and displays the card's response. The smart card does not have an independent display or user input capability. A trusted public authority certifies the public keys of the other authorities.

The basic architecture, and overview of the protocol is shown in Figure 6.1.

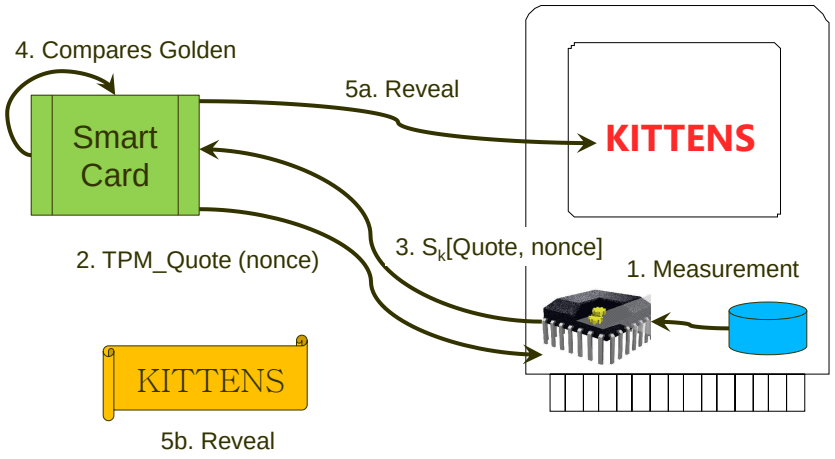


Figure 6.1: Sketch of the architecture and protocol as implemented in a standard PC environment. Steps: 1) trusted BIOS stores software measurements in TPM. 2) Smart card requests a quote from the TPM. 3) TPM replies with digest of PCR values. 4) Smart card looks up PCR values ("golden" measurements) and checks the digest signature. 5a) If PCRs match the smart card golden values, it reveals attestation secret, 5b) user reveals secret on scratch off code and makes sure it matches.

Authorities

We reference the following authorities in our protocol, and they are trusted to do certain things:

- *Platform Authority (PLA)* - procures, configures, deploys, and maintains the voting platform's hardware and software; in the voting context, the election authority acts as the PLA
- *Independent Testing Authority (ITA)* - ensures correctness of voting software and correct compilation of software executables from source
- *Challenge Authority (CHA)* - creates independent challenge strings that voter uses to confirm the attestation decision, manages the smart cards

Other authorities and actors in the election process not central to our protocol include the election authority that oversees the entire process, the legislature that defines voting law, state and national agencies that define voting and cryptographic standards, and candidates and additional stakeholders in the correct outcome of the election.

System Trust

The system actors are trusted to perform their intended duties, such that failure of any one of them can compromise the security claims of our system. In our model, the voter is not trusted, but instead is a consumer of the security provided by our system.

There is no multi-party trust—the system trusts each authority explicitly and independently. Likewise, trust is not transitive among authorities. Last, the voter trusts the challenge authority and the independent testing authority, but does not have to trust the platform.

6.4.2 Component Assumptions

- Voter can compare text strings (suggested by Adida and Neff [4])
- Smart cards are given to voters only on election day
- Voting terminals use version 1.2 TPMs
- The time taken by a certain computing platform to communicate with the smart card and its TPM; and the time for the platform's TPM to perform signature operations, produces time measurements that have a repeatable mean and variance
- The path between a smart card and the PC is a *synchronous* channel as defined by Franklin and Tschantz [38]

6.4.3 Protocol

The protocol is broken up into several protocol *parts*, each part consisting of a sequence of steps. The actor indicated in the part title is the *primary* actor, and performs all the steps in the protocol part. Other actors exchanging information with the primary are explicitly stated.

The commands in this protocol correspond to calls in the *TCG Service Provider Interface (TSPI)*; however, many TSPI calls require numerous arguments and additional setup/takedown commands. We purposely have simplified these elements of the protocol to keep the presentation focused on the overall protocol design. Refer to [19, 98] for API details.

Part 1 - Voting Software Initialization and Certification (ITA)

$$binaries = \textbf{TrustedCompile}(platform\ software \dots) \tag{1.1}$$
$$pcrVals = \textbf{ExtendHash}(binaries) \tag{1.2}$$
$$ITA \rightarrow Public : S_{ITA}(pcrVals) \tag{1.3}$$

1. The ITA receives the platform software from the vendor and reviews the software for correctness according to the design and specification. The platform software consists of all components needed to boot the platform, initiate the voting software, and perform the ballot display, intent capture, and ballot casting operations. The ITA uses a trusted compiler to convert the software source code into executable binaries for the target platform.

2. The ITA computes the expected Platform Configuration Register (PCR) values of the target binaries using cryptographic hashes as defined by the TCG [97]. The PCRs are created by *extension*, an operation that produces a single hash value by ordered concatenation of a sequence of hash values of the individual software components. The $pcrVals$ reflects a "golden measurement" of the platform and systems software.

3. The ITA signs the set of PCR values and posts this for public inspection. For proper security, the ITA certificate must be issued by a trusted root certificate authority.

Part 2 - Platform Load and Key Creation (PLA)

$$\forall i \in platform\ inventory,\ do \tag{2.1}$$
$$plaSRK = \textbf{TakeOwnership}(\dots)$$
$$plaSRKPub[i] = \textbf{GetPubKey}(plaSRK)$$
$$PLA \rightarrow Public : S_{PLA}(plaSRKPub[\,]) \tag{2.2}$$

1. For each platform in the inventory, the PLA takes ownership of the TPM. This creates the *Storage Root Key (SRK)* and exports it to local storage.

2. The PLA obtains the public portions of the SRK from each authorized platform, and signs them.

The PLA loads the voting software onto the platform and distributes the platforms to the precinct managers when appropriate.

Part 3 - Challenge Key Creation (CHA)

$$PLA \rightarrow CHA : plaSRKPub[\,] \tag{3.1}$$

$$ITA \rightarrow CHA : pcrVals$$

$$chaSRK = \textbf{TakeOwnership}(\dots) \tag{3.2}$$

$$quoteSignKey = \textbf{CreateKey}(chaSRK, SIGNING, MIGRATABLE) \tag{3.3}$$

$$\forall i \in platform\ inventory,\ do$$

$$maKey = \textbf{CreateKey}(chaSRK, plaSRKPub[i]) \tag{3.4}$$

$$ticket = \textbf{AuthorizeMigrationTicket}(maKey, REWRAP)$$

$$quoteMigBlob[i] = \textbf{CreateMigrationBlob}(chaSRK, quoteSignKey, ticket) \tag{3.5}$$

1. The Challenge Authority receives the list of public keys from the Platform Authority. It receives the set of expected PCR values from the Independent Testing Authority.

2. The CHA establishes its own platform by taking ownership of its TPM, producing chaSRK.

3. The challenge key, $quoteSignKey$, is created as a migratable signing key. This key will be used by the voting platform to sign the TPM quote challenge.

4. The $quoteSignKey$ is encrypted into special *migration blobs*, one for each platform in the inventory. First, a migration authority key $maKey$ is created out of the SRK public key provided by the Platform Authority, corresponding to the target voting terminal. The migration authority key is merely a fancy wrapper around the target platform's SRK public key, used solely for *wrapping* (encrypting) the quote signing key. Next, a temporary *migration ticket* is created that reflects the intended recipient and usage mode of the $quoteSignKey$ key. The `REWRAP` parameter says to wrap the migration key with the recipient's public key allowing the recipient to load the key only and not forward it on.

5. `CreateMigrationBlob` takes the Challenge Authority's SRK, the quote signing key, and the migration ticket and creates an encrypted blob. The blob can be decrypted only by the TPM that corresponds to $plaSRKPub[i]$ created in Step 2.1.

Part 4 - Smart Card Initialization and Timing Verification (CHA)

$$ITA \rightarrow CHA : pcrVals \tag{4.1}$$

$$CHA \rightarrow Card : \textbf{CardLoadSW}(SelfVerif)$$

$$attTime[card] = \textbf{Time}(SelfVerif,\ nonce) \tag{4.2}$$

$$PLA \rightarrow CHA : \text{sample platform } Plat \tag{4.3}$$

$$CHA \rightarrow Plat : quoteMigBlob[Plat]$$

$$\begin{aligned} attTime[quote] = \textbf{Time}(& \\ & Card \rightarrow Plat : \textbf{Quote}(quoteSignKey, PCRs, nonce) \\ & Plat \rightarrow Card : S_{quoteSignKey}[quoteResp,\ nonce] \\ &) \end{aligned} \tag{4.4}$$

$$CHA \rightarrow Public : S_{CHA}(attTime[\,]) \tag{4.5}$$

---

1. The CHA obtains PCR values from the ITA. Special software for platform attestation and self-verification, $SelfVerif$, is loaded onto the smart cards. The self-verification module is of the type suggested by Seshadri *et al.* [85] that is able to hash itself and a critical software module repetitively, is implemented optimally, and runs in a deterministic amount of time.[5]

2. The CHA verifies integrity of the card by collecting timing data on how long the card takes to process a challenge. CHA verifies that base timing data is the same on all cards of same type, discounting any with significant timing variance. Timing may vary between different models of cards. The CHA verifies the timing against known/published measurements, or against those results obtained in a *cleanroom* setting using a trustworthy instance of the card.

3. The CHA randomly chooses a sample voting platform from the PLA. The CHA should reload a fresh copy of the platform software, or do any other refreshes that are necessary, including taking new ownership of the TPM and repeating protocol parts 2 and 3. The quote migration blob is loaded, and the platform TPM unpacks and loads $quoteSignKey$ from the blob.[6]

4. The card measures the time taken by the platform to receive a quote request, sign the request with the proper key, and return the response. The card also collects timing variance. Last, the platform is completely cleared prior to being returned to the PLA to ensure that the quote signing key can never be loaded again without a new TPM

[5] Different types of cards should be procured from different vendors to ensure diversity, driving up the cost of lifecycle attacks on the cards that could otherwise detect and modify the base timing attestation step. As an alternative, destructive delayering and examination can be performed on random examples of cards to ensure no extra logic gates, or wireless communications mechanisms, are present.

[6] $quoteMigBlob$ could be an associative array, indexed by a hash of the platform SRK public key reported by **GetPubKey**.

owner context. As timing may vary based on the type of platform being used, each type of platform needs its own baseline.

5. The CHA publishes the timing test data and observed timing variances for use by various voting rights groups.

Part 5 - Secret Creation and Card Data Load (CHA)

$$for\ large\ j,\ do:\ \ secrets[j][good|bad] = \textbf{CreateSecrets}() \tag{5.1}$$

$$\textbf{CardLoadData}(pcrVals,\ attTime[\,],\ quoteMigBlob[\,],\ secrets[\,]) \tag{5.2}$$

---

1. The CHA creates a large set of random independent secret string pairs. Each pair includes one string indicating platform attestation success (*e.g.*, "kittens"), and an independent string indicating attestation failure (*e.g.*, "bunnies"). The CHA creates scratch-off tickets encoded with several secret pairs chosen at random, and a serial number uniquely identifying each secret pair.

2. Each smart card is loaded with the "golden measurement" $pcrVals$, the attestation response times and variances for each type of voting platform, the migration blobs of the $quoteSignKey$ for every authorized platform, and the attestation response secrets. The CHA securely distributes the cards and scratch-off tickets to the various voting rights groups.

Part 6 - Voter Attestation (Card)

$$Voter \rightarrow Card : \textbf{Time}(SelfVerif, nonce) \tag{6.1}$$

$$Voter \rightarrow Plat : secSerial \tag{6.2}$$
$$Plat \rightarrow Card : secSerial, platId$$
$$Card \rightarrow Plat : \textbf{LoadKey}(quoteMigBlob[platId], \ldots)$$
$$attTime = \textbf{Time}(\tag{6.3}$$
$$Card \rightarrow Plat : \textbf{Quote}(quoteSignKey, PCRs, nonce)$$
$$Plat \rightarrow Card : S_{quoteSignKey}[quoteResp,\ nonce]$$
$$)$$
$$result = \textbf{Verify}(quoteResp, pcrVals, attTime) \tag{6.4}$$
$$Card \rightarrow Plat : string = \left\{ \begin{array}{ll} secrets[secSerial][good] & \text{if result is good} \\ secrets[secSerial][bad] & \text{otherwise} \end{array} \right\}$$
$$Plat \rightarrow Voter : string \tag{6.5}$$

1. The voter receives a scratch off ticket and a smart card from the voting rights group. The voter (or voting rights group) performs a timing challenge of the smart card, comparing expected times with those published by the CHA, to verify that the smart card is of the correct type and running the correct software. (Other voting details are implied, *e.g.*, signing into the polling location, receiving a ballot from the poll workers, . . .)

2. The voter inserts the smart card into the voting platform. He randomly chooses a secret pair from his ticket and enters the corresponding serial number into the platform. The platform communicates this number to the smart card. The card looks up and sends the correct migration blob to the platform, and the platform loads this key into its TPM.

3. The card begins a timed, quote signing attestation loop with the platform, using a fresh nonce every time. The TPM Quote command takes the handle of a key to sign the quote, and an index specifying which PCRs are to be read and signed; the TPM replies with the values of the PCRs, indicating platform software state, and a signed digest. The smart card records the times for each loop and the variance.

4. The card verifies the quote response against pcrVals and checks that the signature matched $quoteSignKey$, and the card verifies that this loop completed in the expected amount of time with acceptable variance. If these tests pass, the card releases the attestation success string, otherwise it releases the failure string. The revealed string carries no information about the attestation decision made by the card.

5. The voter scratches off the "good" part of the ticket, and confirms the result displayed on the platform. If the strings match, the voter continues interacting with the system. If the strings do not match, the voter may summon a poll worker to reveal the "bad" string to confirm a misconfiguration (or an unknown or no string, revealing an attack or total software failure).

6.5 Protocol and Security Analysis

We describe the security properties of the protocol. We also derive the timing requirements and assumptions necessary to defeat the proxy attack.

6.5.1 Countered Attacks

The protocol defends against attacks on voter privacy. Voter privacy is compromised if the voter interacts with the wrong platform, the right platform running the wrong software,

or a proxy platform covertly interacting with the right platform that has been "kidnapped" to answer the attestation challenge. With these attacks, the adversary learns the voter's intention (violating privacy) while preventing the vote from being recorded on authorized equipment. These attacks can be detected only after the voter verifies his vote on the public bulletin board well after the end of election day.

A further attack is on the smart card itself, causing the smart card to reveal the incorrect attestation string to the voting terminal, hiding malicious software on the terminal. The protocol exposes these attacks in the following ways:

Unauthorized Voting Platform: This scenario occurs when an unauthorized voting terminal, one with a TPM not owned by the platform authority, is placed in the voting booth. The $quoteSignKey$ created in Step (3.3) is wrapped by the public SRK keys of only authorized voting platforms. The SRK is created in Step (2.1) when the PLA takes ownership of the TPM. The CHA's TPM uses the SRK public key from an authorized platform to encrypt the $quoteSignKey$ key during creation of the migration blob $quoteMigBlob$ in Step (3.5), ensuring that only the authorized platform–possessing the corresponding SRK private key–is able to load and use the quote signing key. If an unauthorized platform is asked to sign a quote, it will fail because its TPM's SRK is unable to decrypt the $quoteSignKey$.

Key migration enables the challenge authority to create the key used to sign the quotes. The reader may observe that a TPM quote request can be signed with any signing key, and wonder about the reason for doing key migration. Although there is a system design elegance and added security of the key management process in enabling the challenge authority to create its own keys, the main reason that the CHA creates the keys and not the platform is to ease the burden on the platform authority.

With our design, the platform authority has to do nothing more than certifying and publishing the public key, a minor extra burden to taking platform ownership of the TPM. (A reader skilled in TPM application will notice that the migration steps are severely condensed in the protocol steps; as mentioned, several TPM commands involve lengthy setup and take-down sequences, and none are more demanding than those used in the migration process.)

Authorized Platform, Unauthorized Software: This scenario covers both the case of a minor software component or configuration discrepancy on an otherwise legitimate (known TPM) voting platform, and the case of an adversary completely replacing the software on a legitimate voting terminal. When the platform boots, the *measured launch* process loads and creates cryptographic hashes of the platform software, including the critical operating system, configuration, and voting software binaries and stores them in the TPM PCRs. The TPM Quote request causes the TPM to load the $quoteSignKey$, read and report the PCR measurements stored within, and sign them with the key. Changes to any of the measured launch files will result in differences between the PCR values in the quote response and one or more PCR values in the golden hashes. The smart card uses a copy of the golden hashes determined by the ITA in Step (1.3) to detect these differences in the verification in Step (6.4), and communicates the "bad" string to the platform.

Since the platform always receives a string from the smart card, it has no way to know whether the attestation evidence was confirmed or rejected, and has no choice but to display the string given by the card. In the case of "mostly good" attestation where the platform software is able to follow the protocol correctly, it will display the string given to it which will happen to be the "bad" string. The administrator

can reveal the "bad" string and know that the problem is due to some definite, but honest, misconfiguration or corrupt file. In the case of complete adversary takeover, the malicious software will run the protocol but will have to display a random string to the voter, (or no string at all,) which will match neither the "good" string or the "bad" string with high probability and instantly reveal the corruption.

Proxy Terminal, Kidnapped Authorized Platform: In this scenario, the adversary obtains an authorized voting terminal with a known TPM SRK, places a proxy terminal capable of interacting with a smart card in the voting booth, and connects the two together in such a way that the kidnapped platform is made to answer challenges. During smart card initialization, the challenge authority obtains a sample voting platform and times its quote responses and signatures as indicated in Step (4.4). During attestation, these times are loaded onto the smart card which then confirms the time it takes the platform to respond to the quote request in Step (6.4).

As will be demonstrated in Section 6.5.2, response time not only includes the time needed by the TPM to sign the quote, but also the time that the communication channel takes to deliver the quote message from the smart card to the platform and back again. The proxy case requires an extra hop in the communication path to forward the card challenge to the kidnapped platform and get a response, and we hold that this extra time can be detected by the smart card provided that the timing assumptions in Section 6.5.2 hold. Timing analysis is the key to thwarting the proxy attack.

Smart Card Software Attack: In this scenario, an adversary corrupts the card system software to get it to reveal the "good" string regardless of quote values returned by the TPM. The CHA measures the root of trust of the card software in Step (4.2) prior

to entrusting the card with the secret strings in Part 5. Based on previous research, adversary software cannot fool the timing challenge step in a resource-constrained environment [85], so the secrets are safe. The voter repeats the timing verification in Step (6.1).[7]

6.5.2 Timing Analysis

Our protocol defeats the proxy attack only if fundamental timing constraints on the communications channel between the smart card and the TPM are upheld. The timing model and the resulting constraints are now developed.

We define the following times in our model:

- T_C is the time it takes for the smart card to send a challenge message to the platform CPU (intended for the TPM) over a synchronous channel
- T_P is the time required by the platform CPU to forward the challenge onto the TPM, and send replies back to the card
- T_{TPM} the time taken by the TPM to to exchange messages with the CPU, obtain the PCR values sign the quote and issue the response.
- ϵ is the end-to-end channel noise time resulting from natural variations caused by, for example, environmental effects, system interrupts, and CPU message queuing delays

The expected time to complete one TPM quote challenge in the non-proxy case is:

$$2T_C + 2T_P + T_{TPM} + \epsilon \tag{6.6}$$

[7]The voter does not need to do timing verification of the card if he trusts the chain of custody of the card from the CHA.

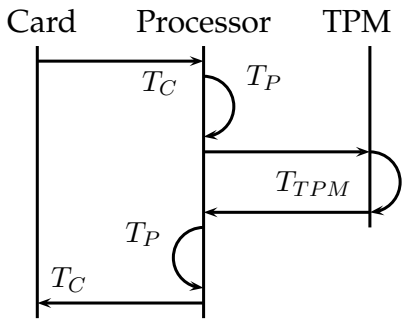


Figure 6.2: Expected Timing Model

Figure 6.2 is the expected timing model. This involves the time for the card to send a challenge to the platform CPU T_C; the time for the CPU to format the proper TPM command, T_P; and the time to send it to the TPM, the TPM to answer the challenge, and reply to the CPU, T_{TPM}. The extra factors on T_C and T_P reflect the return trip times.

To model the proxy attack, we need some additional times:

- T'_P is the adversary's CPU time
- T_N is the time taken to send a message across the adversary's covert network linking the adversary proxy to a kidnapped, legitimate voting terminal

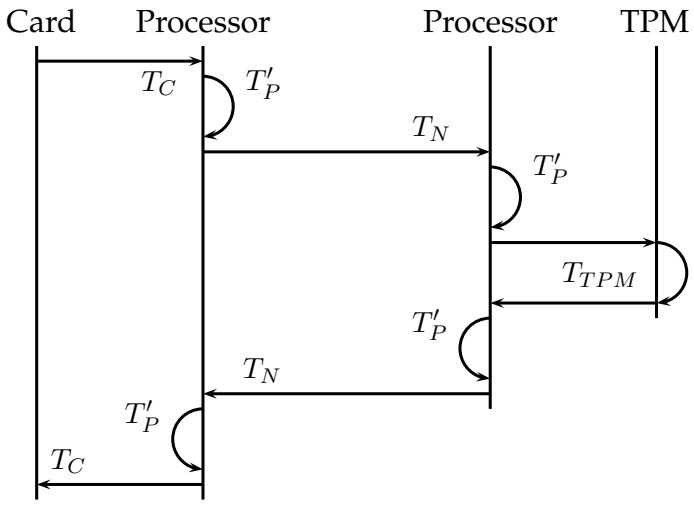


Figure 6.3: Adversary Timing Model

Figure 6.3 is the proxy attack timing model. During the attack, the voter unwittingly inserts his smart card into a proxy machine that accepts the challenge from the smart card, and must transmit it over some covert network to a kidnapped, legitimate platform parked outside the voting booth. The kidnapped platform must answer the challenge, send it back over the network to the proxy which then writes it to the card. Further, we assume that the adversary uses a different, potentially faster processor than we do. This sequence has an expected time of:

$$2T_C + 4T'_P + 2T_N + T_{TPM} + \epsilon \tag{6.7}$$

For the proxy attack to succeed, the time required by the adversary model in equation (6.7) must not exceed the time taken by the expected model in equation (6.6). One strategy is to use a fast processor and network connection, and try to hide in the noise. This

strategy succeeds if inequality (6.8) holds:

$$2T_C + 4T'_P + 2T_N + T_{TPM} \;\leq\; 2T_C + 2T_P + T_{TPM} + \epsilon \tag{6.8}$$

Reducing inequality (6.8) gives:

$$T_N \;\leq\; (T_P - 2T'_P) + \frac{\epsilon}{2} \tag{6.9}$$

If we assume that the time T_P for a CPU to encode or decode a TPM message is no worse than time T'_P to encode/decode a message for the covert adversary network, then we get:

$$T_N + T_P \;\leq\; \frac{\epsilon}{2} \tag{6.10}$$

exposing the proxy attack if either T_N or $T_P = T'_P$ is greater than $\epsilon/2$. (We will determine real-world values of ϵ with further research on timing.)

6.5.3 Attacks Not Countered

Our protocol cannot not defend against the following classes of attacks:

Life Cycle Attacks: If the TPM or smart card hardware are implemented from corrupt designs, or employ malicious extra logic gates, we cannot ensure that the timing attestation phases complete properly, and we cannot believe the results. Most likely, such problems will be detected through vendor diversity or large timing variations, but no known technique can prevent the effects of lifecycle attacks.

Ticket Theft/Reprint: If an adversary steals the stack of secret scratch-off tickets, he can steal the secrets, encode his voting machine with the codes, and reprint and redistribute the scratch cards. This attack requires some skill in reprinting scratch-off cards, as well as a breached chain of custody on the scratch-off tickets.

Insider Attacks: The trusted officials and entities are relied upon to carry out their responsibilities fully. This protocol, and any other protocol, would fail if one or more of the trusted parties failed to do their jobs correctly.

6.6 Alternative Protocols

In this section, we briefly discuss two alternative approaches. The first replaces the smart card with a ”flat file” (*e.g.*, presented to the voting machine on a CD). The second uses physical techniques (*e.g.*, glitter glue in clear epoxy) to authenticate the TPM in the voting machine.

6.6.1 Flat Files

In original discussions, the authors debated the use of smart cards. If defending against the proxy attack were not a requirement, a system of encrypted flat files could be used instead. The *seal* feature of the TPM could enforce the integrity of the platform state by restricting use of a decryption key only to states where the correct software has been booted.

Sealing is a way that a TPM can be made to load a key only if all of the platform configuration registers contain the correct values. Fink’s *et al.* [35] PVB voting protocol relies on TPM sealed keys.

In the flat file protocol, the PLA creates a key sealed to the expected PCR values. Since the sealed key is bound to the TPM SRK, it can only be used on that platform. The CHA receives the public portions of the sealed keys, and uses these to encrypt "good" strings into flat-files—one encrypted secret per sealed key—that are stored on removable media.

In the poll booth, the voter presents the encrypted flat-file to the platform, and selects a challenge serial number from his scratch off card. The platform loads its sealed key into the TPM. If the PCRs match the golden values, the load succeeds, otherwise it fails. On success, the platform then searches the media for the secret encrypted by its sealed key (public portion), decrypts the secret with the TPM, and displays it to the voter. The voter verifies that the decrypted secret matches his scratched off value and proceeds to enter his choices.

The flat file attestation technique is likely cheaper to design, implement, and procure than the smart card attestation protocol and eliminates the scratch-off tickets, but there are significant problems:

1. The CHA must encrypt all secrets with keys from every possible platform that the voter is likely to use. This forces the challenge authority to use smart cards with lots of memory, or somehow manage and organize cards into subsets based on, for example, the assigned precinct.

2. The platform authority must create the sealed keys, not the challenge authority. This is because sealed keys are non-migratable, and therefore must be created on the platform they are to be used on. This has two implications:

   (a) Creating the sealed keys is extra work for the platform authority, and any problems with creating the sealed key will result in a dead platform on election day.

(If the CHA fails to create a migration blob, the voter can just use another smart card)

(b) Letting the platform authority create the keys places requires the voter to trust the platform authority in addition to the challenge authority. If a malicious platform authority administrator creates a key without using the TPM and gives the public portion to the challenge authority, then the malicious system can bypass the TPM and decrypt the attestation secret regardless of platform state. (The CHA creates the key in the smart card protocol, thwarting a malicious platform administrator)

3. The challenges are created ahead of time (the smart card creates a fresh nonce with each attestation loop iteration).

4. The flat-file alternative cannot detect the proxy attack.

6.6.2 Physical Authentication

There are relevant physical authentication techniques that can be used instead of, or in addition to, our protocol, to increase the voter's assurance that he is interacting with a valid voting machine. For example, the machine could be built so that its TPM is clearly visible to the voter–*e.g*, through a window in the front of the machine.

Moreover, the TPM could be secured to the machine by embedding it in clear epoxy glue scattered with glitter. This technique [94] provides both tamper evidence and physical authentication of the TPM: the three-dimensional glitter patterns are unique and essentially impossible to duplicate, and attempts to compromise the epoxy are physically apparent. The epoxy should also contain a clearly visible serial number. Prior to the election, newspapers and websites could publish the glitter patterns for all voting machines.

During voting, each voter could visually examine the TPM and its glitter pattern in his voting machine, comparing the observed pattern to that expected for the given serial number.[8]

This technique provides strong physical assurance to the voter that the voting machine has an authorized TPM. The technique does not guarantee the integrity of other critical aspects of the machine (*e.g.*, display, CPU). Also, the usability of this technique by voters must be verified.

6.7 Future Work

Further work must investigate the timing variance on the communications path—starting with the smart card and ending with the TPM—during TPM quote operations. Timing variance must be determined on how long a fast CPU takes to format a TPM message. Each of these operations should be done against a large number of keys and incorporate differing environmental factors. Further, the attestation approach should be implemented to support usability implications of the attestation approach.

6.8 Conclusions

We have developed a timing- and TPM-based attestation protocol that allows a human to attest the software state of a voting platform prior to entering sensitive information and vote selections into it. This protocol can extend to other sensitive security applications.

[8]For military systems, making the TPM visible holds an additional benefit. In an emergency, a soldier could easily destroy the TPM by firing a gun directly into the TPM, greatly complicating the enemy's task of recovering secrets from the TPM.

Computers may save time but they sure waste a lot of paper. About 98 percent of everything printed out by a computer is garbage that no one ever reads.

Andy Rooney

Chapter 7

On trustworthy receipt printers in the Scantegrity election system

7.1 Introduction[1]

THE Scantegrity system [24] provides *End-to-End (E2E)* voter verifiability. Each voter uses an optical scan paper ballot with special invisible printing in the markable positions. When selecting his candidate, the voter marks his ballot with a special pen filled with reactive ink that reveals a hidden confirmation number in each marked position. The voter can record each number revealed by his selection to check them later on an election website to verify that her vote was recorded and tallied correctly, helping verify election integrity. Each code is random in each contest (sometimes called "race") and each ballot, and therefore does not reveal the corresponding vote. The ballots are handled as traditional ballots, preserving the ability to hand count and perform other traditional election activities associated with optical scan systems. A final tally can be computed from the confirmation

[1]This is joint work and appears in both Richard Carback's and Russell Fink's dissertation with committee and graduate school approval.

numbers, and the system provides a public digital audit trail that allows verification of the election.

A key feature of Scantegrity is enabling results verification without trusting any system component that potentially can change the election results without detection. Practical experience [17, 91], however, points to several problems that occur when voters and election judges interact with the system.

The most prevalent of these problems is that voters have trouble recording the confirmation numbers to make their receipts. Often, voters do not notice that they can make a receipt, or they do not follow the instructions to write down the online verification number and other information so that they can check the receipt online after the election. The codes can also be hard to read, and voters can make mistakes when writing the codes down.

Adding a receipt printer to Scantegrity could solve these problems by automatically generating receipts for voters, substantially improving use experiences with the system. However, a receipt printer potentially introduces a complex trusted component. The Scantegrity authors warned against such a trusted component, and did not propose a design for it. A malicious receipt printer could generate improper receipts that may go unnoticed by voters and cause undetectable changes in election outcomes, and violate voter privacy.

We propose two designs that balance trust and usability, enabling the benefits of a printer while minimizing privacy risk. Rather than trusting the entire receipt printer *platform*–operating system and software of a general purpose computing platform–we place trust in the *Trusted Computing Group (TCG)*'s *Trusted Platform Module (TPM)*, a small, embedded cryptographic processor that safeguards keys from malicious software. Our designs use the TPM to mitigate the problems of malicious software, enabling the benefits of printing without aiding manipulation, integrity, or privacy attacks. Further, the TPM enables our designs to be trusted to maintain secure digital records of the receipts, enabling bulk, third-

party verification of **every** receipt, strengthening the security of the overall Scantegrity approach.

We propose two designs: The first attempts to be as simple as possible, and the other provides more features at the cost of additional complexity. Both of our designs make use of the TPM to protect the confidentiality and integrity of information passing through the receipt printer, offering good usability without compromising privacy or election integrity.

Section 7.2 discusses background information and related work. Section 7.3 presents the functionality and security goals that any receipt printer for Scantegrity must implement, and motivates our design. Section 7.4 provides our two designs, while Section 7.5 evaluates the security of these designs. Section 7.6 presents additional considerations for receipt printers. Section 7.7 discusses a controversial change to Scantegrity that is possible only by using a receipt printer. Section 7.8 concludes the work.

7.2 Related Work

Scantegrity is an E2E voting system [70]. Voting systems that are E2E provide high assurance that the tally is computed properly while maintaining ballot secrecy.

These systems have their origins in work by Chaum [21] in 1981, who first proposed cryptography for the purpose of anonymizing ballots in a verifiable manner. Adida provides a survey of the next two and a half decades of work in this area [6].

The 2005 VVSG proposal [99] initially defined E2E voting systems, and Popoveniuc *et al.* [70] improved on this definition. The first proposals that can be identified as E2E were proposed by Chaum [22] and Neff [63]. Other proposals include: *Prêt à Voter* [26], *Punchscan* [69, 37], the proposal of Kutylowski and Zagórski [59] as *Voting Ducks*, and Simple Verifiable Voting [13] as *Helios* [7] and *VoteBox* [79].

Making usable E2E systems is challenging, and the Scantegrity project has been a leader in this effort. It has been deployed in a Mock election [90, 91] and a real election at Takoma Park [17, 18], and in both cases the Scantegrity team studied the voter experience and concluded that Scantegrity could be used effectively and is well accepted by election officials and voters. They also concluded that making printed receipts for the voters, rather than having voters record confirmation numbers manually, would improve the voter experience and the likelihood of voters verifying their results.

Previous voting designs have used printers. Andy Neff's VoteHere system used a receipt printer as part of its protocol to commit cryptographic codes to the voter [8]. Our system uses a receipt printer for convenience and usability, printing legible codes for the voter, but our printer is not part of the Scantegrity cryptographic commitment protocol.

The printed ballot version of SureVote, invented by David Chaum, prints *sure* codes onto the voter's receipt corresponding to his selections [20]. However, we cannot find any design element for securing the printer or safeguarding the sure codes from surreptitious disclosure by a rogue receipt printer. Our system encrypts the Scantegrity codes using keys managed by the TPM in such a way that ensures only the correctly booted platform software can gain access to the Scantegrity codes.

Fink *et al.* gives a comprehensive overview of the TPM relative to the field of voting in [35]. For other features and references on the TPM, the TCG publishes the main TPM specifications in [97]. Pearson *et al.* give an alternative overview of the TPM and the TCG [68], and Challener *et al.* [19] wrote an excellent practical guide to the TPM for software developers. Developers using the TPM are guided to the TrouSerS software stack and test suite for understanding the programming details [50].

Others have suggested using TPMs for voting. Fink, Sherman and Carback designed a TPM protocol for *Direct Recording Electronic (DRE)* voting that signs the ballot and

voter selections using a key managed by the TPM that provides proof of the correct DRE software state at the time the vote was cast [35]. Arbaugh outlined an on-line TPM-based protocol for attesting systems through a central server [9]. Rössler *et al.* suggested hardware security modules for postal-voting [73]. Paul and Tanenbaum [67] sketched a voting system architecture incorporating TPMs. Although there is interest in using TPMs for voting, no previous approach uses TPMs to design secure receipt printers.

7.3 Requirements

The security of the receipt printer is important, because it can affect the privacy and integrity of the election. As part of the receipt printer designs, we present the high level requirements that any receipt printer for Scantegrity must implement to ensure integrity, authenticity, and confidentiality while improving overall usability, and discuss traceability in the designs.

7.3.1 Functional System Requirements

The functional requirements describe the high level features that any Scantegrity printer must provide:

Printed Scantegrity Codes: The receipt printer will produce Scantegrity receipts, providing the user with the confirmation numbers of the selected candidates and the online verification number for the ballot.

Self-Verifiable Receipts: Anyone should be able to verify that each receipt came from an authorized receipt printer using information printed on the receipt. Further, The voter, and anyone else, shall be able to confirm that the receipt printer booted only authorized software by verifying information printed on the receipt.

Independence: The receipt printer shall not rely on the correct operation of any other component in the system.

Usability: The design should be intuitive to use and able to accommodate accessibility interfaces. A printed format is good for sighted voters, but designs should accommodate disabled voters and speakers of different languages.

Longevity: The same receipt printing equipment should be able to be used in multiple elections over many years.

7.3.2 Security Goals

In addition to basic features, a Scantegrity receipt printer must uphold some security goals or constraints while providing the basic features:

Privacy: The receipt printer should not compromise voter privacy, *e.g.*, by disclosing Scantegrity codes to unauthorized parties or printing information on the receipt that would help correlate a confirmation code to the voter's selection.

Integrity: The receipt printer should not facilitate attacks on the integrity of the election. Further, the receipt printer should not facilitate false challenges to the election integrity. Generally speaking, the printer shall not enable an attacker to impart credibility to a false receipt.

Event Control: The receipt printer shall not be capable of presenting valid cryptographic proof prior to, or proceeding, the authorized election period. This requires strong controls on signature keys.

Information Control: Only authorized platforms shall be entrusted with any sensitive ballot data, *e.g.*, Scantegrity confirmation codes.

There are many ways to meet these requirements and constraints, and a TPM-based approach is merely one way to do this. (A different way might involve provisioning cryptographic material via removable smart cards, but we do not consider such a design because it involves more trusted components than a TPM design.)

7.4 Design

We present two design variations for receipt printing for Scantegrity. The first is a stand-alone image duplicator, and the second is a marked sense translator that requires state and a connection to the PCOS scanner. There are benefits and problems with each, as will be discussed in Section 7.6.

Both variations assume that the voter has completed his ballot prior to requesting a receipt. Neither variation encodes any information onto the receipt that can identify the voter or how he voted.

Both methods record *attestation evidence* onto the receipt, a special cryptographic code generated by the TPM that can prove to the voter that the receipt printer platform booted the correct software. Since the attestation method is common to both approaches, we present its design as a segue from the basic functional design of the image duplicator to the more feature-rich marked sense translator.

Both designs rely on certain features of the TPM. The central TPM features that our designs use include:

- *Platform Configuration Register (PCR)*–storage inside the TPM that securely stores cryptographic hashes of booted printer operating and application software
- *Sealing*–a operation that binds the unwrapping and use of a secret to the identity of the TPM and the software state reflected in the PCRs

- *Monotonic Counters*–non-decreasing counters managed securely within the TPM
- *Quote*–a listing of the PCR values, signed by the TPM
- *Cryptographic Keys*–including the *Attestation Identity Key (AIK)*, a signature key that confirms some *known* TPM without identifying *which specific* TPM, and the *Storage Root Key (SRK)*, the parent of other decryption keys
- *Ownership*–the act of establishing the key hierarchy in the TPM, including creation of the AIK, SRK, and associated keys

Both designs rely on a *core root of trust* for measurement, a process that initiates a series of software measurements made in sequence during platform boot, comprising a *boot chain*. In the trusted computing context, the TPM's PCRs store measurements of all components in the chain enabling sealing, quotes, and other operations. The core root of trust is either some firmware in the *Basic Input/Output System (BIOS)*, or the `AC_INIT` software module present on modern Intel processors (AMD has a similar module).

To simplify the presentation, we describe the functional designs of each alternative in terms of a *use case*, or what the voter sees during interaction. The attestation approach is common to both alternatives, and appears separately.

7.4.1 Image Duplicator

The image duplicator scans the images of all markable positions and prints them in a permuted order onto the receipt. The image duplicator requires no knowledge of the Scantegrity ballot encodings, and no connections to the *Precinct Count Optical Scan (PCOS)* scanner–thus, we call this a *stateless* design. For this reason, the image duplicator is the simpler of the two receipt printer approaches.

Functional Design

The image duplicator consists of an image scanner coupled with a printing device. For the purposes of this design, the image duplicator's scanner and printer are an integrated unit.

A simple use case sequence best describes the major design elements of the image duplicator. The use case follows:

1. The voter completes his Scantegrity ballot in the polling booth.

2. The voter presents his marked ballot to the image duplicator.

3. The image duplicator scans the *markable positions* of the ballot—all of the Scantegrity bubbles, whether marked or not—and also scans the online verification number from a *Two-Dimensional QR barcode (qrcode)*.

4. The image duplicator creates the receipt by printing: (a) images of the marked positions exactly as scanned onto the receipt, but with their order rearranged; and (b) an AIK *digest* of the online verification number[2]

5. The voter verifies the codes from the filled bubbles printed on the receipt with those on his ballot. He also verifies that the online verification number matches his ballot. If there is a discrepancy, he alerts a poll worker and his marked ballot is revoked, consistent with the precinct's practices and procedures.

The voter verifies the integrity of the image duplicator software using the attestation protocol described in Section 7.4.2.

[2] A digest consists of some plaintext values and a hashed representation of them signed by some key, the AIK in this case.

Details

The critical design details of the image duplicator include how the markable positions–the bubbles–are identified and scanned, and how the bubble images are presented and used.

Markable Positions The image duplicator identifies markable positions within individual contests using x, y offsets from preprinted alignment marks detected on the ballot. The Scantegrity ballot uses dark circles to identify the qrcode and the markable positions to the PCOS scanner [25], and the image duplicator will reuse this feature.

The image duplicator scans an image of the entire area of each markable position. A suggested resolution for the scan is 150 dots per inch and 8-bit grayscale (256 levels of gray), sufficient for resolution of the revealed Scantegrity codes. The image duplicator does not attempt to determine the filled state of the corresponding bubble, but merely captures the image as marked on the ballot.

Scanned Bubble Printing The image duplicator groups the images of the scanned markable positions by contest, and sorts the images within each contest by average pixel value. The average pixel value is computed over an 8-bit grayscale representation of the image. Images of blank bubbles will appear after images of partially marked and fully marked bubbles, respectively. For each contest, the image duplicator prints a contest indicator, *e.g.*, contest 1, and the scanned bubble images for that contest, onto the receipt.

Receipt Usage The voter compares the verification code images printed on his receipt with those reflected on his ballot, to make sure that the verification codes are correctly and intelligibly recorded. A potential implementation issue is whether the grayscale scanning and print features can render the verification codes legibly onto the receipt;

having the voter verify his codes at this stage can help determine whether the receipt will be useful to the voter later.

An example receipt is shown in Figure 7.1.

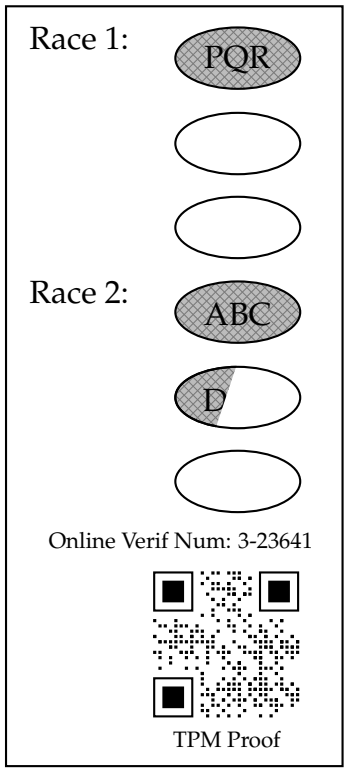


Figure 7.1: Receipt from image duplicator. Images of scanned bubbles are printed in order of average pixel density, grouped by contest. Notice that partial marks also appear. (Overvotes and undervotes are detected by the PCOS later in the voting process.)

Before presenting the marked sense translator, we break for a moment to show the attestation design, as it is the same for the image duplicator as it is for the marked sense translator.

7.4.2 Receipt Printer Attestation Protocol

Our claim is that the authorized software will print the correct information and will safeguard the voter's privacy by not disclosing his preferences. We ensure this claim by giving the voter a way to confirm that the receipt printer loaded the correct software at boot time. To verify the state, the TPM gives the voter cryptographic evidence of the platform software state, and in this way, the TPM is said to *attest* the state of the platform to the voter. The attestation design uses the TPM to report boot-time measurements of the platform state securely.

The two design alternatives for attestation are *pre-scanning* attestation, in which the voter verifies the software prior to scanning her ballot or entrusting the printer with other private information, and *post-scanning* attestation in which the voter verifies the platform software state after scanning his ballot, using the attestation proof printed on it. While the pre-scanning variation is better for privacy, the post-scanning variation is more practical, and still can detect misconfiguration or rogue software before the voter leaves the polling location.

Attestation protects voter privacy and receipt authenticity–it cannot guarantee election integrity, and therefore is not a substitute for the voter verifying the confirmation codes on his receipt against those on his ballot. The voter, or a trusted third-party voting rights group, should use at least one type of attestation to prove to the voter that he is interacting with a correctly configured receipt printer.

We discuss the individual phases of attestation: system initialization, election day initialization, voter attestation, and termination.

System Initialization

The *Election Authority (EA)* performs some initialization to enable platform attestation:

1. The EA takes *ownership* of the receipt printer TPM. Ownership establishes the key hierarchy in the TPM. It is done only once, and is good for as long as the EA owns the equipment.

2. The EA determines the set of expected PCR values by measuring software components of the approved build for the receipt printer.

3. The EA commands the TPM to create the AIK used for signatures, sets an authorization password on the AIK, and binds the AIK to the expected PCR values. The EA certificate authority signs the AIK public key certifying that it was created by an approved TPM.

4. The TPM creates a monotonic counter bound to the authorization password and PCR values, similar to the AIK. Note, this counter is only useful in the marked sense translator as explained in Section 7.5.

5. The EA creates a delegation of the TPM `ownerClear` command used to destroy the TPM key hierarchy, and sets a special tear-down password on the delegation.

The EA publishes the AIK public key, the AIK certificate, and the PCR values, but keeps the AIK and tear-down passwords a secret until election day. The critical secret component is the AIK private key, and it is exposed only inside of the TPM.

Although several AIK keys should be created per platform to prevent an adversary from correlating a usage pattern to a particular platform, we assume a single AIK to simplify the above narrative.

Election Day Initialization

1. At the start of election day, the EA distributes the AIK password to the poll workers.

2. The poll workers boot the receipt printer. The printer performs a *trusted boot* process where each booted component cryptographically measures each subsequent component, storing the measurements in the TPM's PCRs. The poll workers enter the AIK password into the receipt printer; the password and correct PCR values enable the use of the AIK.

3. The receipt printer platform prints the digest of the monotonic counter value, signed with the AIK.

4. (Poll workers do other general initialization, *e.g.*, test scans to check scanner and printer connections, alignment integrity, ink and paper levels, etc.)

Election Day Attestation

1. The voter marks his ballot during voting, and takes it to the receipt printer.

2. In ***pre-scanning attestation***, the voter uses a smart card to verify the receipt printer's integrity using the technique discussed in [36]:

   (a) The voter obtains a smart card from a trusted third party.[3]

   (b) The smart card generates a random number called a *nonce*, and sends it with a quote request to the receipt printer.

   (c) The receipt printer requests a `TPM_QUOTE`. The TPM fetches its PCRs and signs them and the nonce with the AIK, producing the quote.

[3]The trusted third party can verify the smart card using a self-signing timing technique, as explained in [36, 39, 85].

(d) The receipt printer returns the quote to the smart card, along with the AIK key identifier. These data are called the *attestation proof*.

(e) The smart card checks its keys dictionary for the AIK. If found, it verifies the PCR values in the quote reply, and validates the signature.

(f) (Steps 2b to 2e can be repeated several times by allowing the smart card to time the responses to detect a proxy/oracle attack [36].)

(g) Upon successful verification, the smart card reveals an *attestation secret* to the receipt printer–a word, phrase or number that means the platform software is correct–and the printer reveals this secret to the voter.[4]

(h) The voter confirms the attestation secret with the trusted third party, and alerts a poll worker if he cannot confirm the secret.

3. The voter releases ballot details and the ballot online verification number to the receipt printer (design-specific, either by direct scanning or by PCOS transmission).

4. The receipt printer issues a `TPM_QUOTE` as in Step 2c, but using the online verification number as the nonce. The TPM responds per Step 2d.

5. The receipt printer prints the Scantegrity receipt, including the Scantegrity codes and the online verification number and attestation proof and additional design-specific proof as required. The proof is printed as a qrcode for ease of use.

6. In ***post-scanning attestation***, the voter uses a trusted device to read the qrcode, look up the AIK public key, and verify the attestation proof. The trusted device can be a *Personal Digital Assistant (PDA)* or cell phone, or can be a separate computer

[4]The implementation dictates how the secret is revealed: if the printer has a display, it shows the smart card secret on it; if not, it prints the secret on a blank sheet of paper.

maintained by a trusted third party, possibly a voting rights group located in the polling place.

7. If the attestation proof fails verification, the voter alerts a poll worker to take remedial action.

Election Termination

At the end of the election period, the platform signs the final value of the monotonic counter with the AIK. When the EA releases the `ownerClear` password, the poll workers enter it causing the TPM to erase its key hierarchy so that the AIK private key never can be used again. Although not part of attestation, the poll workers can retrieve digital archives of the receipts at poll close time, to publish for verification by third-parties.

Cryptographic Keys Summary

The AIK (and the *Verification Codes Secret (VCS)*, used by the marked sense translator for confidentiality described in Section 7.4.3) have several properties shown in Table 7.1.

Operation	AIK	VCS
Purpose	Prove platform booted correct software	Protect Scantegrity verification codes from disclosure
Type	Asymmetric (public/private)	Asymmetric (public/private)
Owner	EA	EA
Key Creation	At platform initialization	Just after platform initialization
Import/Export	Public/plaintext	Sealed blob
Distribution	One or more unique AIK per platform	One VCS shared over all platforms, wrapped by individual platform SRK
Parent	Internal TPM *Endorsement Key (EK)*	TPM SRK public
Key Use	Sign online verification number and PCR values	Decrypt Scantegrity verification blob
Authorization	PCR values, election day password	PCR values, election day password

Table 7.1: Properties of keys in the Scantegrity receipt printer.

We will now show an additional use of the TPM in the second variant of the receipt printer described below.

7.4.3 Marked Sense Translator

The marked sense translator connects directly to the PCOS scanner. It receives mark sensed positions from the PCOS, translates the positions into the Scantegrity codes that should be revealed on the ballot, and prints the codes onto a paper receipt. Unlike the image duplicator, the marked sense translator requires knowledge of the Scantegrity codes for each cast ballot, making it a *stateful* design. It also reports a count of the number of receipts printed, to support auditing.

Functional Design

Figure 7.2 gives an overview of the marked sense translator in operation. A use case describes the high level design:

1. The voter completes his Scantegrity ballot in the polling booth, then presents his ballot to the PCOS for scanning.

2. The PCOS scans the marked bubbles and ballot ID from the ballot. It interprets the marked bubbles as *selections*. The PCOS sends the selections and ballot ID to the marked sense translator. Other data, such as overvote or undervote details compliant with *Help America Vote Act (HAVA)* requirements, may be transmitted also [1].

3. The marked sense translator securely retrieves the Scantegrity verification codes for each selection, and prints them onto a paper receipt. It also prints an AIK digest of both the voter's codes and the monotonic counter value. The TPM increments its counter.

4. The voter verifies that the Scantegrity codes on the receipt match those on his ballot.[5] If there is a discrepancy, he alerts a poll worker and his cast ballot is retrieved and revoked, consistent with the precinct's *Policy and Procedures (PAP)*.

The voter verifies the integrity of the marked sense translator software using the attestation protocol described in Section 7.4.2.

[5] Section 7.4.3 discusses practical ways of comparing codes.

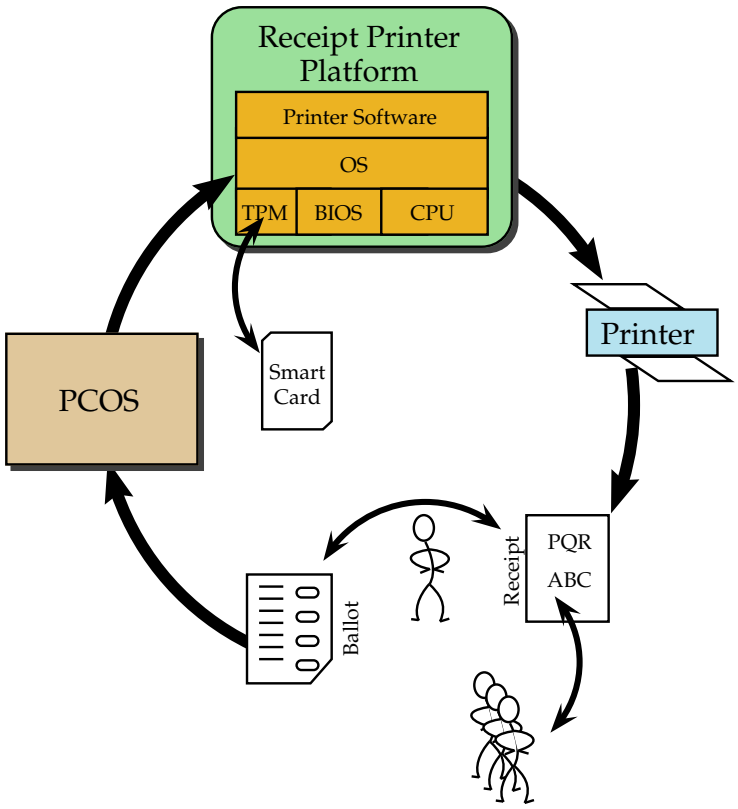


Figure 7.2: Overview of the marked sense translator. The voter submits his ballot to the PCOS that sends the marked positions, optional encrypted ballot definition, and online verification number to the marked sense translator. The voter optionally uses a smart card to verify the platform. The software uses the TPM to reveal the Scantegrity codes, prints the codes and attestation proof onto the receipt. The voter compares the receipt to the ballot. Anyone may verify the integrity of the receipt and the marked sense translator with the attestation proof on the receipt.

Details

The critical design details of the marked sense translator include contents of the receipt, protection of the Scantegrity verification codes, and modifications required of the PCOS.

Receipt Contents As with the image duplicator, the marked sense translator groups the revealed codes by contest. It randomly orders the codes within each contest. Unlike the

image duplicator, the marked sense translator does not scan the ballot, and therefore it does not report codes of partially marked bubbles, those not sensed by the PCOS.

Connection to PCOS The PCOS is connected to the marked sense translator using a data cable. The data sent to the marked sense translator include (a) ballot ID; (b) contest designations; (c) marked positions by contest (*e.g.*, "contest 1, position 1 of 3 is marked"); and (d) optional indication of overvoting or undervoting per contest.

The PCOS must enforce a well-defined message interface format to protect it from a corrupt marked sense translator that may send ill-formed messages to the PCOS. A one-way data cable *may* mitigate this threat, but it might break any message that requires acknowledgment from the marked sense translator.

Scantegrity Codes Retrieval The marked sense translator uses the private portion of its VCS–bound to the platform PCRs–to decrypt the codes corresponding to the scanned ballot. The codes for each ballot are encrypted uniquely for every platform by the EA, and are indexed by ballot ID. Further, the EA signs the codes blob proving authenticity of the codes to the marked sense translator.

Required PCOS Modifications The PCOS must be modified to supply the ballot ID and sensed marked positions to the image duplicator. These data are part of the content already retained by the PCOS.

Cryptographic Key Summary

The EA creates the VCS and encrypts it with the public portions of the SRK of each platform. A common password restricts the VCS use until election day, and the key is sealed to the receipt printer PCR values. The EA loads the encrypted VCS blobs onto the platforms

during system initialization. Table 7.1 in Section 7.4.2 summarizes the properties of the VCS keys.

Design Enhancements

Some enhancements to the basic design include how to display the scanned ballot, and how to transport the Scantegrity codes efficiently and securely to the marked sense translator.

Ballot Image Display

The voter needs a way to cross-check the Scantegrity codes on the receipt with those on his ballot. Unlike the image duplicator, the marked sense translator lacks a scanner and relies on the PCOS for its information. As a consequence, once the voter submits his ballot to the PCOS scanner, it is no longer available to her.

Although a straightforward solution is to have the voter handwrite a few spot check codes from his ballot onto a piece of scratch paper to check against the receipt, this is problematic and defeats the purpose of the printer. Therefore, we suggest two usable design alternatives that let the voter see his votes to ensure that the PCOS sensed the correct marks: a *lever cast* mechanism on the PCOS, and an integrated graphical ballot display.

The lever cast mechanism is a glass display case attached to the output end of the PCOS scanner. The glass case makes the scanned ballot fully visible–but unalterable–until the voter pulls a physical lever to release it into the ballot hopper. After receiving his receipt, the voter compares it to his ballot under the glass, and if the codes match, he pulls the lever to drop the ballot into the hopper; if there is any problem, the ballot can be retrieved and revoked.

The integrated high-resolution display renders a graphical image of the ballot with the Scantegrity codes filled into the marked ovals. It recreates the ballot using information sent

from the PCOS. The display shows (a) individual contest and possible choices as printed on the physical ballot; (b) blank bubbles for unselected choices (including undervoted contests); (c) filled bubbles showing Scantegrity codes (including overvoted contests); and (d) the ballot ID number and other information that will be printed on the receipt.

After receiving his receipt, the voter scrolls through the display and ensures that (a) the displayed ballot reflects his intent; and (b) the Scantegrity codes on the receipt match those on the display.

The display could incorporate magnification or an audible interface for improved accessibility.

For voter privacy, the image on the display can never be captured or printed, just as we do not photocopy the Scantegrity ballot itself.

Transporting State with the Ballot

One challenge is transporting the ballot state–the Scantegrity verification codes–to the marked sense translator so that it can print the correct verification codes. A simple design is for the EA to encrypt the verification codes for every ballot with the VCS, and load these onto the platforms' persistent storage prior to election day. Unfortunately, this reduces flexibility by requiring extra work prior to the election.

A smarter design would have the PCOS read the encrypted verification codes from the ballot itself, and transmit these to the marked sense translator. The qrcode printed on the Scantegrity ballots can encode up to 2,953 binary bytes, enough for about 1,400 individual 3-digit codes. Transporting the codes with the ballot reduces pre-election work, and requires only a single chain of custody for both the physical ballot and its digital representation.

7.4.4 Policy and Procedures

With both designs, some general procedures must be followed to ensure the security of the system. At a high level,

- All authorized receipt printers must be in visible locations, *e.g.*, no printer can be carried off to an undisclosed area during the election by malicious poll workers
- The poll booth must be free of cameras, covert microphones or speakers, networking equipment or anything that can allow communication or observation between an external attacker and the voter
- Reasonable physical security of the printers must be enforced prior to the election, heading off physical attacks against the scanning mechanism or the TPM[6]

The security of the system relies on a majority of poll workers knowing, and correctly enforcing these policies and procedures, regardless of design choice.

7.4.5 Requirements Traceability

The features of both the image duplicator and the marked sense translator map well to the requirements we defined before in Section 7.3.1. Both print Scantegrity codes on the receipts. The operations of both are independent of other components in the system, such that failures in other system components do not cause either receipt printer to fail *e.g.*, reveal voter privacy or fail to print codes. Both designs offer good usability; although the marked sense translator is better for disabled voters, the image duplicator could be made to enlarge the scanned codes making it easier for sighted voters with visual impairments. Both designs

[6]Software injection is "fair game" as it is done much more quickly than microprocessor delayering attacks

can be used many times over many elections: this is trivial for the image duplicator, and the marked sense translator needs only a new batch of encrypted codes for each election.

Anyone can verify the software integrity of both designs using information printed on the receipts. The marked sense translator offers completely verifiable receipts because it signs the codes that it printed on the receipt with the measurements of the platform software, binding software integrity to the codes.

The security goals in Section 7.3.2 are met similarly by both designs. Privacy is maintained by virtue of the platform running the correct software. Integrity, including false challenge prevention, day-before attacks, and sensitive data handling are features all handled by the TPM through sealing decryption and signature keys to PCRs and also to special passwords that are revealed at the start and end of election day.[7]

7.5 Security Analysis

We rely on the assumptions of TPMs for our system security, but in this section we evaluate the total impact a receipt printer will have on election security. We analyze what happens when these assumptions are violated by an attacker to get control of a receipt printer, and what types of attacks he could perform.

7.5.1 Threat Model

We limit our model to attacks that utilize a rogue receipt printer during the election. See [24] for a more general security analysis of Scantegrity and voting, and [35] for an analysis of TPM protocols for voting. In the election context, our adversary could be an insider, a

[7] During early voting in the 2010 Nevada state election, a registrar official allegedly allowed a couple to vote after the polls were closed based on their stated preferences [41].

foreign government, a minority of corrupt poll workers, one or more of the contestants, or a coerced or paid voter. We consider four general categories of attack:

1. *Manipulation Attacks*, where an adversary attempts to manipulate the election result.

2. *Identification Attacks*, where an adversary attempts to identify voter choices and violate election privacy.

3. *Disruption Attacks*, where an attacker wishes to prevent certification of the election undetectably.

4. *Discreditation Attacks*, where an attacker imbues sufficient doubt in the public's perception of Scantegrity's worth.

We do not consider denial of service attacks explicitly, although *disruption attacks* are similar. A denial of service attack is applicable to any voting system and is difficult to prevent but easy to detect. Covert disruption attacks can be considered a special case of denial of service, where the adversary undetectably delays certification of the election when it suits his purpose.[8]

Note that many attacks involve procedural elements that are not easily captured by a cryptographic description, so our intent is not to establish and prove security properties in a formal cryptographic model. Instead, we provide an informal analysis of the underlying security goals in our design. Because our analysis is limited to attacks involving the receipt printer, we consider their designs successful if they do not increase the ability of an adversary to carry out successful attacks undetectably.

[8]Undetectability is central to disruption. While attestation failure of the receipt printer may delay the election, at least the EA knows the problem and the printer unit in question. A successful disruption attack will offer little clue as to where the problem is.

7.5.2 Assumptions

Assumptions listed here are limited to those made on the receipt printer and how it should be used in an election scenario. Some assumptions, such as unreadability of the codes, are required by Scantegrity and not just the receipt printer.

1. TPM Integrity–the TPM correctly implements the TCG specifications and does not leak information it is entrusted with. In particular, the TPM safeguards the AIK private key, the monotonic counter, and the ownership authorization secrets.

2. Supporting Hardware Integrity–the platform BIOS and `AC_INIT` module, if applicable, initiates the measured boot process correctly and stores initial measurements in the TPM's PCRs. The scanner, printer, and integrated display mechanisms operate correctly.

3. Software Correctness–platform software is free of critical bugs or supply chain trap doors, and does not become compromised during runtime.

4. Trusted Actors–(a) EA correctly manages ballot creation and privacy, the *Certificate Authority (CA)*, the public bulletin board, and the AIK and ownerClear secrets; and (b) a majority of poll workers follow correct PAP (further described in Section 7.4.4).

5. Voter Actions–(a) voters check the receipt's online confirmation number prior to releasing the ballot; and (b) for the image duplicator, voters check the Scantegrity confirmation codes in addition to the online number.

6. Ballot Security–(a) obtaining information about a voter's preferences given only the confirmation codes is intractable; (b) scanning equipment cannot read unmarked codes printed in invisible ink; and (c) confirmation numbers and printed mark positions do not contain subliminal information that can influence voters.

We believe that these assumptions are reasonable, standard in the literature, and correspond to existing assumptions about Scantegrity and the TPM as used for voting.

7.5.3 Manipulation Attacks

Manipulation attacks are when the adversary tries to change the outcome of the election without anyone noticing. They succeed in Scantegrity only when not enough voters verify their confirmation codes on the public bulletin board. A malicious receipt printer could cause the voter to verify the *wrong* codes against the bulletin board, if the voter fails to verify all the details of her receipt, enabling an efficient manipulation attack.

For example, a malicious receipt printer could perform a *chain printing* attack where it caches and reprints a valid scanned image and ballot online confirmation number saved from a previous ballot, leading the voter to verify the wrong ballot on the bulletin board. In this way, chain printing would "pigeonhole" Scantegrity verifications preventing certain ballots from ever being verified, while misleading voters into verifying other ballots multiple times. The PCOS could flip votes of the unverified ballots, altering the outcome of the election.

Chain printing is detected using the image duplicator if the voter verifies his online verification number and enough of his confirmation codes before he releases his ballot to the PCOS. In the marked sense translator, a voter or an independent third-party verifies the AIK signature on the receipt and carefully tracks previously seen online verification numbers, posting observed numbers to a public bulletin board. In both the image duplicator and the marked sense translator, if the voter verifies the attestation proof on the receipt, he detects malice in the receipt printer restricting ballot modifications to a malicious PCOS and detecting a discrepancy on the public bulletin board–precisely the attack case that Scantegrity is designed to prevent.

Further, keeping digital archives of the receipts enables independent verification of every receipt very quickly and easily, showing an actual improvement in election integrity by combining Scantegrity with a receipt printer.[9]

In all of these cases, the attack is detectable by any attentive voter that is affected by it. It would not be clear, however, if the attack was malicious or simply an equipment failure, though the equipment could be flagged for inspection and not used for the rest of the voting period.

Other defenses include audited ballot scanning where auditors vote and run test ballots through all receipt printers to check correctness of the receipt generation. An audit like this would provide an advantage in the marked sense translator design, because the auditors would be testing the scanner hardware (not software) at the same time.

Because Scantegrity is only concerned with counting the ballots that were cast correctly, it could be vulnerable to ballot stuffing attacks in the event that officials fail to count the number of voters accurately. A receipt printer cannot prevent election day ballot stuffing because the PCOS, not the printer, is the authoritative vote tallying device. In particular, the image duplicator has no connection to the PCOS and cannot track any part of the process. However, the marked sense translator acts as a trustworthy counter and can act as a backup to official voter counts for detecting polling place stuffing attacks. If the marked sense translator is malicious, the signatures and attestation evidence will fail verification. If the PCOS is malicious but the marked sense translator is good, two scenarios play out that involve the monotonic counter:

[9]When using DREs, many voters do not check the printed receipts when voting [30], and this might be true for the receipt produced by the printer; therefore, automatic checking of digital archives is extremely useful.

1. PCOS fails to communicate to the marked sense translator: the EA observes a discrepancy in the number of printed receipts compared with the number of digital vote records included in the PCOS storage.

2. PCOS communicates to the marked sense translator to print a receipt: adjacent, legitimate voters notice a discontinuity in the counter value stamped on their receipt. Also, poll workers notice a higher receipt print count than the number of actual voters at the end of the day.[10]

   Attackers could produce valid receipts for stuffed ballots in this case, but the discrepancy in voters versus printed receipts would have called all the results of the affected poll into question, including the attackers' stuffed ballots.

As mentioned, the image duplicator cannot help detect election day ballot stuffing attacks because it is not integral to the ballot casting process. Same for the marked sense translator if there are some independent PCOS not connected to one.

7.5.4 Identification Attacks

An identification attack with a malicious receipt printer would allow an adversary to determine how a voter has voted simply by recording the voters choices and the online verification number. Alternatively, to avoid having to be accessed by the adversary later, it could print hard to notice "markers" on the receipts to indicate the selection by the voter, through a subliminal channel in the signature or by some graphical marker on the receipt. It would be difficult to detect or prevent these privacy attacks.

[10]To prevent "behind the curtain" ballot stuffing attacks, poll workers note which authorized machines participated in the election [35]. Additionally, TPM *tick stamps* could stamp the "time" on the receipt for correlation with observed arrival times of authorized voters.

If assumption 6c is violated, an attacker could use the information on the receipt provided by the receipt printer to coerce or identify the selections made on that ballot. An attacker would have to violate this assumption by subverting the printing authority or the Scantegrity trusted workstation. This attack is somewhat orthogonal to the receipt printer, but it could succeed in the image duplicator design because the images on those receipts are not posted publicly. An attack could be discovered in a marked sense translator design because all of the information is publicly posted to a bulletin board, allowing auditors to verify the pseudorandomness of the Scantegrity confirmation numbers.

The image duplicator design may be susceptible to tampering that would reveal ballot selections. If voters can modify the ballot to skew the alignment detection, a slight angle and/or offset may make it possible to determine the order the codes appeared on the ballot. The implementation should be strict regarding the alignment detection and reading the online verification number, and should not print a receipt if it cannot align the ballot image.

7.5.5 Disruption and Discreditation Attacks

The manipulation attacks described in Section 7.5.3 can be used as disruption attacks. In general, simply having misbehaving equipment can affect the perception of trustworthiness in the election, but it does not inherently prevent certification of the election.

An undetected malicious receipt printer could print and sign illegitimate receipts *e.g.*, chain printing. This would delay certification of results, as the paper record would have to be consulted to determine the legitimacy of false claims.

The marked sense translator design, because it has access to all the confirmation numbers of the scanned ballot, could enable an attacker to submit false challenges for every ballot it sees. But for this to work, assumption 1 must be violated. This attack would certainly disrupt certification of the election, and the electorate might, rightly, believe that the privacy

of the election had been compromised upon discovery of such a large number of legitimate complaints, leading to disruption and discreditation of Scantegrity. Another variation would reveal additional codes to voters (as in an overvote). Voters may not notice in the polling place, and could legitimately complain that the receipt does not match the online record after the election.

Receipt forgery is not possible with the marked sense translator, but cannot be prevented easily by the image duplicator. The marked sense translator signs the ballot state (Scantegrity verification codes) with its AIK preventing forgery. Since the attestation evidence is printed on the receipt, anyone can verify the receipt for authenticity and correctness. Unfortunately, the image duplicator cannot prevent forgery, because the only state it has are the scanned representations of the bubble images. Rescanning a physical artifact may lead to different digital values of the page, a fact used to great benefit by certain randomization techniques *e.g.*, [64]. Image processing algorithms may be able to "bin" the scanned ballot images into a small set of discrete values, making it difficult to forge the receipt without violating the signature.

7.6 Discussion

Both receipt printer designs provide two distinct advantages over using the underlying election system only:

1. Usability is improved when voters can use the printers to produce receipts automatically. Automatic receipts save voters time and energy, making each voter more likely to produce and check the receipt after the election. It may decrease the amount of time each voter spends in the polling site and increase the flow of voters through the

polling place. Voters who accidentally undervote or overvote the ballot can also be better informed about what happened via the receipt printer.

2. Security is improved through signed, publicly authenticatable receipts. Such receipts weaken several attacks that may exist in the underlying system. A signed and authenticated receipt gives a voter stronger evidence of recording errors than simple knowledge of a code. A receipt that does not authenticate properly is proof of an equipment issue that should be investigated. The receipt printer can also now provide proof of an under- or overvote, preventing a ballot from being invalidated or allowing change of intent.

The main disadvantage of the printer designs are that, in general, they become an attractive target for violating voter privacy during the election and must be implemented carefully. Because they are polling place devices, even under the assumption that the central authority can store them properly, an attacker could have up to several days to tamper with the machines when they are deployed right before the election, including carrying out costly physical presence attacks [72].

Assuming the hardware is secure, an attacker could not fake a receipt. This makes an attack on the printer costly and unlikely. If it could be done, however, it is unclear when fake hardware would be noticed and that could also pose a denial of service problem.

Only one voter needs to find a discrepancy for an attack on the printer to be discovered. If the intent of the attacker is to attack privacy or election integrity, then a 50% chance of detection makes it a very high risk approach. If the intent is simply disruption, these attacks would be effective; however, there are many simple and presumably less costly disruption attacks (*e.g.*, distributed denial of service on the bulletin board mechanism where receipts are posted).

7.6.1 Comparison of the Designs

The image duplicator design is simple compared to the marked sense translator. The marked state of each bubble is ignored, avoiding complex algorithms to decide whether a bubble is filled, blank, or partially filled. Printing the bubbles in order of average pixel intensity, within contests, is straightforward and gives enough position permutation to protect how the voter voted.

The image duplicator design has the advantage of being intuitive to voters. Because it is simple and a completely separate component with one function, it is easier for a voter to understand what it is supposed to do. Also, to a security conscious voter, it may appear like a more secure design decision to deploy it as a separate component. Making it a separate component may allow the receipt printer—a completely new device—fit in better with a voter's mental model of elections ("Oh, this thing is supposed to give me a receipt to improve security, I get it"). Attaching the receipt printing to an existing component with a well known function may make it harder to comprehend what is happening.

As a standalone independent component, the image duplicator could be used as an option and would therefore have minimal impact on voter flow through the polling site. However, the independence could introduce problems with voter flow if confused voters tried to scan their receipt or other materials instead of their ballot.

The marked sense translator design makes the process easier for the voter. It eliminates having to scan the ballot twice. It allows the voter to verify that the PCOS sensed and recorded the ballot marks correctly, and if there is a problem on the ballot (*e.g.*, overvotes) it can be shown to the voter and poll workers immediately, catching problems early.

The marked sense translator design allows for better accessibility features. Because the selected confirmation numbers are known, they can be provided easily with audio. Since the Scantegrity codes reveal nothing about the voter's preferences, the marked sense translator

could also broadcast the confirmation numbers over a standard interface, allowing voter-controlled devices to get signed digital copies of the receipt on their own trusted devices (this is very useful for blind voters). The image duplicator design could be outfitted with a similar interface, but—barring a optical character recognition algorithm—is limited to providing the image data only.

The marked sense translator design, in general, has a smaller digital footprint than the image duplicator design. A 3-digit confirmation number can be expressed in 3 bytes or less, depending on the number of symbols used to create the confirmation number. Images generally require much more space to store. Thus, the marked sense translator design produces smaller receipt files than the image duplicator design. This can turn out to be a advantage to an auditor who needs to verify a large number of receipts.

Both designs add cost to the election. The image duplicator design incorporates scanning hardware that would be more expensive initially than the marked sense translator. Additionally, the marked sense translator requires changes to the PCOS to support the one-way data transfer interface. The long-term cost of operations of both should be minimal, compared to the value of the security and usability benefits.

7.6.2 Design Tradeoffs and Other Considerations

Election officials could deploy both types of receipt printer, as a way of verifying the receipt printers independently from each other, and independently from the PCOS scanner. This could catch problems early, and perhaps a scanner with the marked sense translator attached could be used in addition to other scanners, with only some voters using the marked sense translator for the better accessibility capabilities and reduced overall cost (and reduced likelihood of catching ballot stuffing attacks).

A one-way cable between the PCOS and marked sense translator provides extra protection to safeguard the PCOS from a rogue marked sense translator. There is no other security property, and therefore a two-way cable could be used as long as the PCOS implements a rigorous interface definition. Enabling the PCOS to hang onto ballots after scanning helps the poll workers to retrieve the ballot easily in the event of an attestation failure or a receipt check problem. It is important that the cast lever be mechanical in this situation (even if the mechanical lever merely moves a computer controlled motor into contact with the ballot) because the scanner should not be able to make the decision to cast without action from the voter. Otherwise it would not be possible to determine if a complaint from a voter is legitimate.

In the marked sense translator, the confirmation codes decryption key cannot be used if the marked sense translator platform booted the wrong software, but transporting the encrypted confirmation numbers on the ballot is better for security because not all codes would be known to all TPMs. This limits exposure in the event that an adversary gains control of a TPM. Using an AIK instead of any other key keeps knowledge of the *specific* TPM a secret, while proving that the signature came from *some* valid TPM, thwarting voter identity attacks.

7.7 Extensions

The marked sense translator, described in Section 7.4.3, produces verifiable codes to the voter in a highly readable format—including audible formats, not possible with the image duplicator—and it catches any marked position sensor errors in the polling place (or vote flips!) committed by the PCOS. These features make the marked sense translator the best

design for usability, accessibility, and security, and is the best way to employ a printer in the Scantegrity architecture.

Thinking further, since we entrust the marked sense translator platform with the Scantegrity confirmation codes, can we just print the codes on the receipt instead of printing them redundantly on both the receipt **and** the ballot? If we could, we can eliminate invisible ink altogether. We explore this possibility through an alternative design.

7.7.1 Design

The basic design prints the codes on a receipt, becoming the way that the voter receives his Scantegrity confirmation codes. A key design consideration is that if we *late-bind* the codes to the ballot using the receipt, we must enable the voter to verify that the codes actually correspond to his choices on his ballot—that is, he must be able to tell that the PCOS sensed his marks correctly. Recall that in the marked sense translator, the voter submits his ballot to the PCOS and no longer has access to his ballot, except to look at it behind glass in a lever cast configuration.[11] There are two cases to consider.

Random Order Ballots

Some jurisdictions allow random candidate reordering on the ballots, such that the order of candidates within each contest varies from ballot to ballot (like it does with Punchscan [69]).[12] In the random reordered ballot case, the marked sense translator could print a copy of the ballot, omitting candidate names but printing the confirmation codes in the correct position, or listing an index of marked positions and corresponding codes. The voter

[11] The image duplicator is a stand-alone design, and therefore cannot verify the PCOS.

[12] This avoids the *primacy* problem, where undecided voters choose the first listed candidate more than any other. See Miller [61].

confirms that the marked positions are correct before casting. This approach resembles Chaum's visual cryptography idea, described in [22].

For example, consider three choices in a hypothetical race for president, Bush, Buchanan, and Gore, in that order, with the voter selecting Buchanan on the ballot. The PCOS scans the ballot and transmits the selection to the marked sense translator, which prints the confirmation code for Buchanan plus an index of "middle selection made." The voter checks his ballot to ensure that Buchanan is the middle choice in the race. The voter checks the election bulletin board later for his Buchanan code.

Fixed Order Ballots

Some jurisdictions object to random ordering, arguing that massively duplicated, fixed-order ballots are less confusing to voters and are cheaper to produce than randomly ordered ballots. In the fixed-order ballot case, the marked sense translator can detect PCOS errors only if it can reveal the voter's selections safely, without violating his privacy. To do this, it creates a temporary replica of the voter's ballot, and either displays it on a graphical (or audible) terminal, or prints it on a separate sheet that must be destroyed before the voter leaves the scanning station. The voter checks the replica of his ballot, and ensures that it reflects his intent before releasing his ballot for casting.

7.7.2 Choosing A Design

Given the two choices for receipt printer; a potential need for the voter to verify the PCOS before leaving the polling location; and the need to minimize costs, we present a decision tree in Figure 7.3 that describes the election experience, from the voter's perspective, for various configurations.

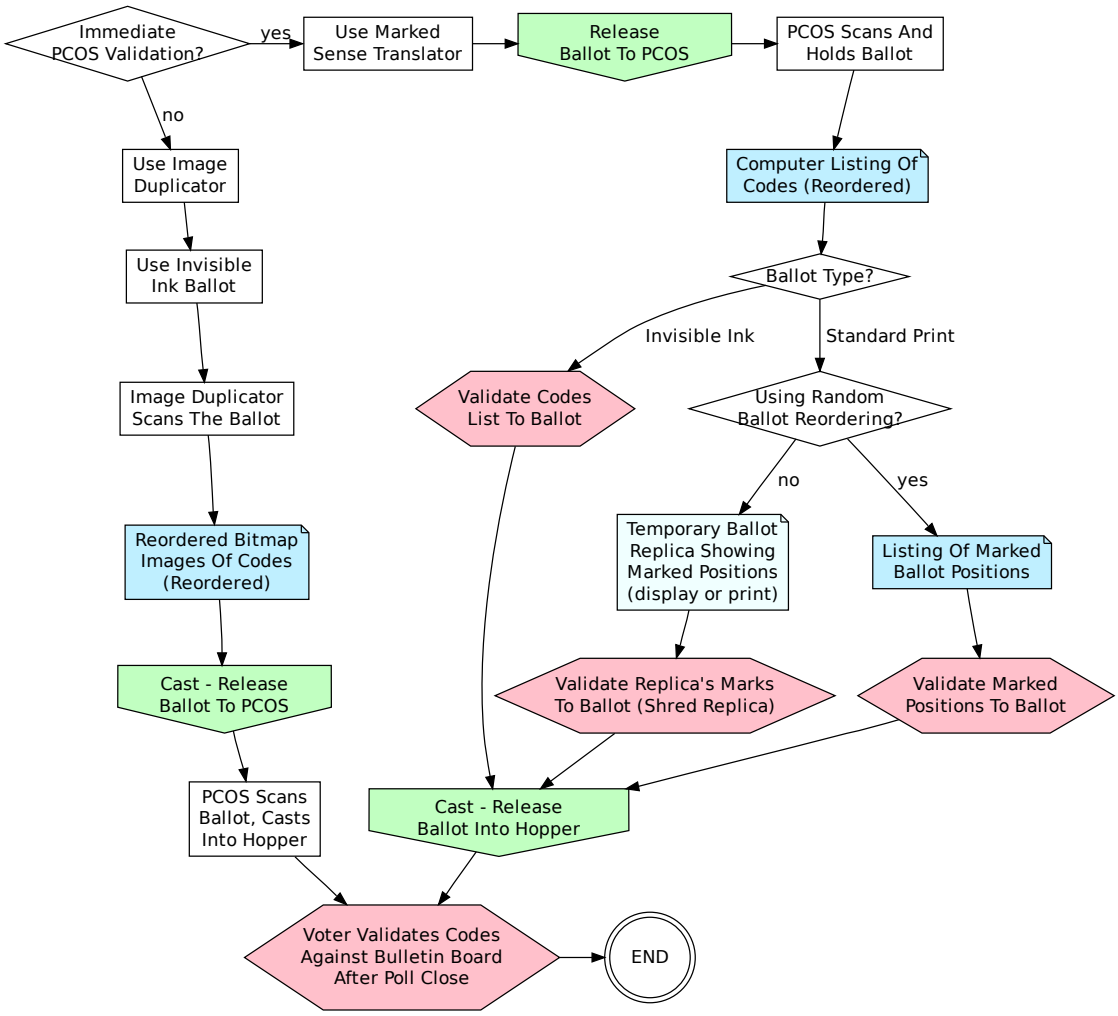


Figure 7.3: Election officials may have to choose between the marked sense translator and the image duplicator depending on their specific election requirements and procedures: is it necessary to detect PCOS scanning errors in the polling location, or can it wait until later? Are normal ballots used or are invisible ink ballots used? Is random ballot ordering permitted, or must all candidates appear in the same order on every ballot?

7.7.3 Concerns and Benefits

Critics may argue that a replica ballot, used in the fixed-ordered ballot case, threatens voter privacy. Researchers including Sherman *et al.* studied vote verification devices, and concluded that voter privacy is at risk with many of these systems [92]. Since the marked sense translator is a vote verification technology, similar problems could exist. For instance, the voter verifies the replica ballot in the scanning booth, and the poll worker potentially could see the ballot and learn the voter's preferences.

A display covered by a special hood could mitigate the poll worker threat, but curiously, this threat exists already in the lever cast mechanism—the same security controls for lever cast should be applied to the replica ballot.

By trusting the marked sense translator as the place where the voter first sees his codes, we eliminate invisible ink and all the problems that go with it. We enable officials to deploy traditional optical scan PCOS ballots that are cheap and easy to produce while still being able to catch errant or malicious PCOS behavior as soon as it happens. Voters might require less assistance with a traditional ballot than an invisible ink one, reducing the burden on officials while preserving voter privacy. Additionally, trusting the digital print subsystem with the codes makes it easy to add audible accessibility devices, removing the need for custom Scantegrity ballot markers and verifiers. By late binding the codes to the ballot, we also eliminate chain of custody attacks against the ballot. Of more concern is that the prototype form of invisible ink darkens over time revealing the codes—and revealing the inherent risks of protecting privacy through chemistry and material processes, risks that are much less with traditional printing processes. In short, the marked sense translator can revolutionize Scantegrity by enabling alternative interfaces and eliminating the most costly and problematic part of preparing the ballots.

7.8 Conclusions

While the image duplicator design has some advantages over the marked sense translator design due to its simplicity, the lack of accessibility features and its additional costs are something that should be considered when procuring the receipt printer. The security of the marked sense translator design, while more complex, is not adversely affected by being closely coupled with the PCOS, and the advantages it offers in accessibility appear to outweigh its disadvantages. Because they are independent and offer complementary benefits, both designs could be deployed.

Three key design decisions greatly improve the security properties over other design choices. First, the attestation function allows any voter, election official, or observer to verify the integrity of the software running on the printer. Second, carefully limiting exposure of secret information by using only data printed on the ballot in each design—instead of giving each receipt printer access to all ballot codes—greatly reduces the privacy risk involved when using a receipt printer. Last, any attacks on the integrity by producing false receipts are difficult because the voter verifies the generated receipt against the voted ballot in each design. As there is no way to cheat undetectably (as seen in DRE review screens), it is highly likely such attacks using the printer will be caught.

In conclusion, while the marked sense translator and the image duplicator design have different advantages and disadvantages, both can provide a usable and viable way to automatically generate receipts in the Scantegrity election system. Using the TPM as a trusted base helps us verify that the platforms are using the correct software, the receipts are genuine, and voter privacy is maintained.

We can only see a short distance ahead, but we can see plenty there that needs to be done.

Alan Turing

Chapter 8

Conclusions

We have analyzed how to apply *Trustworthy Computing (TC)* to voting systems. We have shown that TC has a place among all voting systems that use computer technology, including End-to-End systems that are software independent. Our protocols are based on common, inexpensive cryptographic hardware that ships with every modern desktop, laptop, and server computer configuration, making them readily accessible by systems designers for implementing in new systems. By using Trustworthy Computing, we enable the safe addition of accessibility interfaces, automatic vote verification by third party voting rights groups, improve the usability of the election systems for voters. Adding TC to Scantegrity, for instance, will lead to much more interest in adopting it for large municipalities.

In a greater sense, although integrity may be maintained by software-independent methods, we have shown that privacy can only be ensured through secure hardware. Thus, Trustworthy Computing brings a somewhat forgotten security property back into voting, and into related fields such as healthcare and defense, as well.

In performing this work, we encountered some initial opposition from the *End-to-End (E2E)* community. It was then that we realized that TC can enable privacy and accessibility

in these high integrity voting systems. The balance involves knowing what to trust and for what purpose. Systems designers face a number of trade-offs when creating secure systems. We feel that the ability to provide the same system for every voter safely, with alternative interfaces, is worth trusting a single, simplistic hardware component to protect signature keys in software-intensive platforms. Further, no system in a large democracy can function efficiently without software. Since the need for software in voting is never going away, we must find a way to use it more safely, and *Trusted Platform Modules (TPMs)* is a viable and low-cost alternative that exists today.

We also encountered opposition to TC, and the TPM in particular. Free-use advocates decry that the TPM is used primarily for digital rights management, supporting the corporations by denying certain freedoms to the end-user. Our work has positioned the TPM as a protector of voter rights, transforming the role of the TPM from "mall cop" to "bodyguard." Our work uses TC altruistically showing that the TPMs can help guarantee fair elections, upholding freedom for all.

Looking forward, the software / *Direct Recording Electronic (DRE)* and E2E communities must work closely together to answer fundamental questions: where can hardware security best be applied? What controls must be put in place to detect effects of malicious hardware? Can trusted hardware enable the use of software in ways that reduce perceived system complexity in cryptographic voting systems, increasing acceptance and understanding by voters and systems procurement officials? Removing trust from software by using inexpensive hardware may have profound implications beyond those that we have discussed, and may lead to more simple, maintainable, and understandable election systems.

Bibliography

[1] 42nd Congress of the United States of America. The Help America Vote Act of 2002 (HAVA). United States Public Law 107-252, 2002.

[2] M. Abadi, C. Burrows, C. Kaufman, and B. Lampson. Authentication and delegation with smart-cards. *Science of Computer Programming*, 21(2):93–113, 1993.

[3] B. Adida and C.A. Neff. Ballot casting assurance. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*. USENIX Association, 2006.

[4] B. Adida and C.A. Neff. Efficient Receipt-Free Ballot Casting Resistant to Covert Channels. In *EVT/WOTE'09: Proceedings of the USENIX Electronic Voting Technology Workshop/Workshop on Trustworthy Elections*, Berkeley, CA, USA, 2009. USENIX Association.

[5] B. Adida and R.L. Rivest. Scratch & Vote: self-contained paper-based cryptographic voting. In *Proceedings of the 5th ACM workshop on Privacy in electronic society*, pages 29–40. ACM, 2006.

[6] Ben Adida. *Advances in Cryptographic Voting Systems*. PhD thesis, MIT, August 2006.

[7] Ben Adida. Helios: Web-based open-audit voting. In *Proceedings of the 17th Usenix Security Symposium (USENIX Security 2008)*, pages 335–348. USENIX Association, July 2008.

[8] J.M. Adler, W. Dai, R.L. Green, and C.A. Neff. Computational details of the votehere homomorphic election system. In *Proc. Ann. Intl Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2000.

[9] William A. Arbaugh. The real risk of digital voting? *Computer*, 37(12):124–125, 2004.

[10] AustralianPolitics.com. Hanging Doors, Pregnant Chads And Dimples: Florida Recount Proceeding! AustralianPolitics.com, November 12, 2000. Available at http://australianpolitics.com/news/2000/00-11-12.shtml, 2000. Last accessed November 3, 2010.

[11] Adam Aviv, Pavol Černy, Sandy Clark, Eric Cronin, Gaurav Shah, Micah Sherr, and Matt Blaze. Security evaluation of ES&S voting machines and election management system. In *EVT'08: Proceedings of the conference on Electronic voting technology*, pages 1–13, Berkeley, CA, USA, 2008. USENIX Association.

[12] J. Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, page 14. USENIX Association, 2007.

[13] Josh Benaloh. Simple verifiable elections. In *EVT'06: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, 2006. USENIX Association.

[14] Debra Bowen. Top-to-bottom review. Available at http://www.sos.ca.gov/elections/elections_vsr.htm, 2007. Last accessed June 23, 2010.

[15] Kevin Butler, William Enck, Harri Hursti, Stephen McLaughlin, Patrick Traynor, and Patrick McDaniel. Systemic issues in the Hart InterCivic and Premier voting systems: reflections on project everest. In *EVT'08: Proceedings of the conference on Electronic voting technology*, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.

[16] R. Carback, D. Chaum, J. Clark, J. Conway, A. Essex, P.S. Herrnson, T. Mayberry, S. Popoveniuc, R.L. Rivest, E. Shen, et al. Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy. *IEEE Transactions on Information Forensics and Security*, 4:4, 2009.

[17] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy. In *19th USENIX Security Symposium*, Washington, DC, USA, August 2010. USENIX Association.

[18] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, Poorvi L. Vora, and Bimal Sinha. Exploring Reactions to Scantegrity: Analysis of Survey Data from Takoma Park Voters and Election Judges. Pending Publication, 2010.

[19] D. Challener, K. Yoder, R. Catherman, D. Safford, and L. Van Doorn. *A practical guide to trusted computing*. IBM press, Upper Saddle River, NJ, 2007. ISBN 978-0132398428.

[20] David Chaum. SureVote. Available at http://surevote.com. Last accessed November 3, 2010.

[21] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, 1981.

[22] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security and Privacy*, 2(1):38–47, 2004.

[23] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity II: end-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *EVT'08: Proceedings of the Conference on Electronic Voting Technology*, pages 1–13, Berkeley, CA, USA, 2008. USENIX Association.

[24] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II End-to-End Verifiability by Voters of Optical Scan Elections Through Confirmation Codes. *IEEE Transactions on Information Forensics and Security: Special Issue on Electronic Voting*, 2009.

[25] David Chaum, Aleks Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan Sherman, and Poorvi Vora. Scantegrity: End-to-end voter-verifiable optical-scan voting. *IEEE Security and Privacy*, 6(3):40–46, 2008.

[26] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical, voter-verifiable, election scheme. Technical Report Series CS-TR-880, University of Newcastle Upon Tyne, School of Computer Science, December 2004.

[27] Compuware Corporation. Direct Recording Electronic (DRE) Technical Security Assessment Report, State of Ohio, Office of the Secretary of State. Available at http://www.sos.state.oh.us/sos/hava/compuware112103.pdf, 2003. Last accessed Mar 15, 2008.

[28] Coq. The coq proof assistant. Available at http://coq.inria.fr. Last accessed October 29, 2010.

[29] Europa Press. 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). Available at http://bit.ly/dpeOM0 (short). Last accessed October 29, 2010.

[30] Sarah P. Everett. *The Usability of Electronic Voting Machines and How Votes Can Be Changed Without Detection*. PhD thesis, Rice University, May 2007.

[31] S.P. Everett, K.K. Greene, M.D. Byrne, D.S. Wallach, K. Derr, D. Sandler, and T. Torous. Electronic voting machines versus traditional methods: Improved preference, similar performance. 2008.

[32] Ariel J. Feldman, J. Alex Halderman, and Edward W. Felten. Security analysis of the diebold accuvote-ts voting machine. In *EVT'07: Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, pages 2–2, Berkeley, CA, USA, 2007. USENIX Association.

[33] R. Fink. A Statistical Approach to Remote Physical Device Fingerprinting. In *Military Communications Conference, 2007. MILCOM 2007. IEEE*, pages 1–7. IEEE, 2008.

[34] R. Fink and A.T. Sherman. Combining end-to-end voting with trustworthy computing for greater privacy, trust, accessibility, and usability (summary). In *Proceedings of the National Institutes of Technology (NIST) workshop on end-to-end voting systems*, October 13-14 2009.

[35] Russell A. Fink, Alan T. Sherman, and Richard Carback. TPM meets DRE: Reducing the trust base for electronic voting using trusted platform modules. *IEEE Transactions on Security and Forensics*, 4(4):628–637, 2009.

[36] Russell A. Fink, Alan T. Sherman, and David C. Challener. A human attestation protocol for trustworthy electronic voting: bootstrapping trust using TPMs, smart cards, timings, and scratch-off codes. Unpublished manuscript, June 2010.

[37] Kevin Fisher, Richard Carback, and Alan Sherman. Punchscan: Introduction and system definition of a high-integrity election system. In *Preproceedings of the 2006 IAVoSS Workshop on Trustworthy Elections (WOTE 2006)*, Robinson College, Cambridge, United Kingdom, June 2006. Available at www.punchscan.org/papers/fisher_punchscan_wote2006.pdf.

[38] J. Franklin and M.C. Tschantz. On the (Im) possibility of Timed Tamper-Evident Software in (A) synchronous Systems. 2008.

[39] Ryan W. Gardner, Sujata Garera, and Aviel D. Rubin. Detecting code alteration by creating a temporary memory bottleneck. *IEEE Transactions on Security and Forensics*, 4(4), 2009.

[40] General Dynamics C4 Systems. General dynamics' high assurance systems. Available at http://www.gdc4s.com/highassurance, 2010. Last accessed June 23, 2010.

[41] Jeff German. Vote allegations keep authorities busy. Las Vegas Review-Journal, Nov. 2, 2010. Available at http://www.lvrj.com/news/vote-allegations-keep-authorities-busy-106505448.html. Last accessed November 3, 2010.

[42] Stephen N. Goggin, Michael D. Byrne, Juan E. Gilbert, Gregory Rogers, and Jerome McClendon. Comparing the auditability of optical scan, voter verified paper audit trail (VVPAT) and video (VVVAT) ballot systems. In *EVT'08: Proceedings of the conference on Electronic voting technology*, pages 1–7, Berkeley, CA, USA, 2008. USENIX Association.

[43] D. Grawrock. Dynamics of a Trusted Platform: A building block approach. 2009.

[44] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009.

[45] B. Harris, D. Allen, and L. Alexander. *Black box voting: Ballot tampering in the 21st century*. Talion Pub., 2004.

[46] Paul S. Herrnson, Richard G. Niemi, Michael J. Hanmer, Benjamin B. Bederson, Patrick G. Conrad, and Michael W. Traugott. *Voting technology: The not-so-simple act of casting a ballot*. Brookings Institution Press, 2008.

[47] G.J. Holzmann. The model checker SPIN. *Software Engineering, IEEE Transactions on*, 23(5):279–295, 2002.

[48] P. Hudak, S. Peyton Jones, P. Wadler, B. Boutel, J. Fairbairn, J. Fasel, M.M. Guzmán, K. Hammond, J. Hughes, T. Johnsson, et al. Report on the programming language Haskell: a non-strict, purely functional language version 1.2. *ACM Sigplan Notices*, 27(5):1–164, 1992.

[49] H. Hursti. Diebold TSX evaluation: Critical security issues with Diebold TSx Black Box Voting. Available at http://www.bbvdocs.org/reports/VVBreportIIunredacted.pdf, 2006. Last accessed March 15, 2008.

[50] IBM Corporation. The Trusted Computing Software Stack (TrouSerS) software library. Available at http://trousers.cvs.sourceforge.net/viewvc/trousers/, 2008. Last accessed June 23, 2010.

[51] IBM Corporation. The IBM 4758 PCI cryptographic coprocessor. Available at http://www-03.ibm.com/security/cryptocards/pcicc/overview.shtml, 2010. Last accessed November 3, 2010.

[52] IBM Corporation. Software TPM emulator. Available at http://ibmswtpm.sourceforge.net/, 2010. Last accessed June 23, 2010.

[53] D.W. Jones. A brief illustrated history of voting. *University of Iowa Department of Computer Science. Available at http://www.cs.uiowa.edu/~jones/voting/pictures/*, 2003.

[54] Andreu Riera Jorba, Jos Antonio, Ortega Ruiz, and Paul Brown. Advanced security to enable trustworthy electronic voting. In *In Proceedings of the 3rd European conference on e-Government*, pages 377–384, 2003.

[55] J. Kelsey. Strategies for software attacks on voting machines. In *NIST Workshop on Threats to Voting Systems*, 2005. Available at http://vote.nist.gov/threats/papers/strategies_for_software_attacks.pdf.

[56] J. Kelsey, A. Regenscheid, T. Moran, and D. Chaum. Attacking Paper-Based E2E Voting Systems. *Towards Trustworthy Elections*, pages 370–387, 2010.

[57] T. Kohno, A. Broido, and KC Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, pages 93–108, 2005.

[58] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. *IEEE Symposium on Security and Privacy*, page 27, 2004.

[59] Mirosaw Kutyowski and Filip Zagrski. Verifiable internet voting solving secure platform problem. In Atsuko Miyaji, Hiroaki Kikuchi, and Kai Rannenberg, editors, *Advances in Information and Computer Security*, volume 4752 of *Lecture Notes in Computer Science*, pages 199–213. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-75651-4%5F14.

[60] Peter A. Loscocco, Perry W. Wilson, J. Aaron Pendergrass, and C. Durward McDonell. Linux kernel integrity measurement using contextual inspection. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 21–29, New York, NY, USA, 2007. ACM.

[61] J.M. Miller and J.A. Krosnick. The impact of candidate name order on election outcomes. *Public Opinion Quarterly*, 62(3):291, 1998.

[62] National Security Agency / Central Security Service. The high assurance platform program. Available at http://www.nsa.gov/ia/programs/h_a_p/index.shtml, 2010. Last accessed June 23, 2010.

[63] C. A. Neff. Practical high certainty intent verification for encrypted votes, 2004.

[64] Landon Curt Noll, Robert G. Mende, and Sanjeev Sisodiya. Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system. US Patent, Mar 1998. 5,732,138.

[65] L.D. Norden and Brennan Center for Justice. *The machinery of democracy: Voting system security, accessibility, usability, and cost*. Brennan Center for Justice at NYU School of Law, 2006.

[66] L.D. Norden and E. Lazarus. *The Machinery of Democracy: Protecting Elections in an Electronic World*. Brennan Center for Justice at NYU School of Law, New York, NY, 2007.

[67] Nathanael Paul and Andrew S. Tanenbaum. Trustworthy voting: From machine to system. *Computer*, 42(5):23–29, 2009.

[68] S. Pearson and B. Balacheff. *Trusted computing platforms: TCPA technology in context*. Prentice Hall PTR, 2003.

[69] Stefan Popoveniuc and Ben Hosp. An introduction to punchscan. In *Proceedings of the 2006 IAVoSS Workshop on Trustworthy Elections*, 2006.

[70] Stefan Popoveniuc, John Kelsey, and Andrew Regenscheid. Performance requirements for end-to-end verifiable elections. In *EVT/WOTE'10: Proceedings of the*

Electronic Voting Technology Workshop/Workshop on Trustworthy Elections, page 16, Berkeley, CA, USA, 2010. USENIX Association/IAVoSS/ACCURATE.

[71] RABA Innovative Solution Cell. Trusted agent report: Diebold AccuVote-TS voting system. *State of Maryland General Assembly, Department of Legislative Services*, 2004. Available at http://www.raba.com/press/TA_Report_AccuVote.pdf. Last accessed March 15, 2008.

[72] Jordan Robertson. Security chip that does encryption in pcs hacked. USA Today, Feb 8, 2010. Available at http://usat.ly/9gdlNR. Last accessed October 25, 2010.

[73] Thomas Rössler, Herbert Leitold, and Reinhard Posch. E-voting: A scalable approach using xml and hardware security modules. *e-Technology, e-Commerce, and e-Services, IEEE International Conference on*, 0:480–485, 2005.

[74] Joanna Rutkowska. Introducing blue pill. Available at http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html. Last accessed February, 2009.

[75] Mark Ryan. Introduction to the TPM. Available at http://www.cs.bham.ac.uk/~mdr/research/projects/09-TrustedComputing, 2008. Last accessed June 23, 2010.

[76] A.R. Sadeghi, M. Selhorst, C. Stüble, C. Wachsmann, and M. Winandy. TCG inside?: a note on TPM specification compliance. In *STC '06: Proceedings of the first ACM Workshop on Scalable Trusted Computing*, pages 47–56. ACM, 2006.

[77] SAIC Corporation. Risk assessment report: Diebold Accuvote-TS voting system and processes (unredacted). Available at http://www.brad-blog.com/?p=3731, 2003. Last accessed Mar 15, 2008.

[78] D. Sandler, K. Derr, and D.S. Wallach. VoteBox: a tamper-evident, verifiable electronic voting system. In *Proceedings of the 17th conference on Security symposium*, pages 349–364. USENIX Association, 2008.

[79] Daniel R. Sandler, Kyle Derr, and Dan S. Wallach. VoteBox: a tamper-evident, verifiable electronic voting system. In *Proceedings of the 17th Usenix Security Symposium*, 2008.

[80] Luis F. G. Sarmenta, Marten van Dijk, Charles W. O'Donnell, Jonathan Rhodes, and Srinivas Devadas. Virtual monotonic counters and count-limited objects using a TPM without a trusted OS. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, pages 27–42, New York, NY, USA, 2006. ACM.

[81] Scantegrity. The Scantegrity website. Available at http://www.scantegrity.org, 2010. Last accessed November 3, 2010.

[82] Science Applications International Corporation. Risk assessment report: Diebold AccuVote-TS voting system and processes (unredacted). Available at http://www.brad-blog.com/?p=3731. Last accessed March 15, 2008.

[83] Seagate Corporation. DriveTrust technology: A technical overview. Available at http://www.seagate.com/docs/pdf/whitepaper/TP564_DriveTrust_Oct06.pdf. Last accessed February, 2009.

[84] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems.

In *Proceedings of the twentieth ACM symposium on Operating systems principles*, page 16. ACM, 2005.

[85] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. SWATT: SoftWare-based ATTestation for embedded devices. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 272–282. IEEE, 2004.

[86] P. Sevinç, M. Strasser, and D. Basin. Securing the distribution and storage of secrets with trusted platform modules. *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, pages 53–66, 2007.

[87] Michael I. Shamos. Paper v. electronic voting records-an assessment. In *Proceedings of the 14th ACM Conference on Computers, Freedom and Privacy*, 2004.

[88] Michael I. Shamos. Voting as an Engineering Problem. *BRIDGE-WASHINGTON-NATIONAL ACADEMY OF ENGINEERING-*, 37(2):35, 2007.

[89] Alan T. Sherman. Election Tracker: A new tool for greater election transparency. In *The First University Voting Systems Competition (VoComp)*, 2007. Rump session talk.

[90] Alan T. Sherman, Richard Carback, David Chaum, Jeremy Clark, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Bimal Sinha, and Poorvi L. Vora. Scantegrity Mock Election at Takoma Park (summary). In *Workshop on End-to-End Voting Systems*, Washington, DC, USA, October 2009. National Institute of Standards and Technology.

[91] Alan T. Sherman, Richard Carback, David Chaum, Jeremy Clark, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily

Shen, Bimal Sinha, and Poorvi L. Vora. Scantegrity Mock Election at Takoma Park. In *EVOTE2010: The 4th International Conference on Electronic Voting*, Bregenz, Austria, July 2010. E-Voting.CC.

[92] A.T. Sherman, A. Gangopadhyay, S.H. Holden, G. Karabatis, A.G. Koru, C.M. Law, D.F. Norris, J. Pinkston, A. Sears, and D. Zhang. An examination of vote verification technologies: Findings and experiences from the Maryland Study. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, page 10. USENIX Association, 2006.

[93] A. Silberschatz, HF Korth, and S. Sudarshan. *Database System Concepts (2005)*. New York: McGraw-Hill, 2005.

[94] G.J. Simmons. *Contemporary cryptology: The science of information integrity*. IEEE press, 1992.

[95] Mario Strasser. A software-based TPM emulator for Linux. *Department of Computer Science, Swiss Federal Institute of Technology, Zurich*, 2004.

[96] Mario Strasser, Heiko Stamer, and Jesus Molina. Software-based TPM emulator (software). Available at http://tpm-emulator.berlios.de/, 2010. Last accessed June 23, 2010.

[97] Trusted Computing Group. TCG TPM specification version 1.2, revision 103. Available at https://www.trustedcomputinggroup.org/specs/TPM, 2008. Last accessed on Mar 15, 2008.

[98] Trusted Computing Group. The TCG Software Stack. Available at http://www.trustedcomputinggroup.org/developers/software_stack, 2009. Last accessed Sep 1, 2009.

[99] United States Election Assistance Commission. The 2005 Voluntary Voting System Guidelines. http://www.eac.gov/testing_and_certification/2005_vvsg.aspx, December 2005.

[100] J.N. Wand, K.W. Shotts, J.S. Sekhon, W.R. Mebane, JR, M.C. Herron, and H.E. Brady. The butterfly did it: The aberrant vote for Buchanan in Palm Beach County, Florida. *American Political Science Review*, 95(04):793–810, 2002.

[101] Ka-Ping Yee. *Building reliable voting machine software*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 2007. Adviser-Wagner, David and Advisor-Hearst, Marti.