

**Cryptographic End-to-end
Verifiability**
for
Real-world Elections

by

Aleksander Essex

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2012

© Aleksander Essex 2012

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this dissertation we study the problem of making electronic voting trustworthy through the use of cryptographic end-to-end (E2E) audits. In particular, we present a series of novel proposals for cryptographic election verification with a focus on real-world practicality. We begin by outlining fundamental requirements of E2E election verification, important properties for a real-world settings, and provide a review of previous and concurrent related work. Our research results are then presented across three parts.

In the first part we examine how E2E election verification can be made more procedurally familiar to real-world voters and election administrators. We propose and implement an E2E add-on for conventional optical-scan based voting systems, and highlight our experiences running an election using this system in a United States municipality.

In the second part we examine how E2E election verification can be made more conceptually and procedurally simple for election verifiers/auditors. We present a non-cryptographic E2E system based on physical document security assumptions as an educational tool. We extend this system to a cryptographic setting to show how the procedures of cryptographic election verification can be completed with relatively tiny software code bases, or by using common-place programs such as a desktop spreadsheet. We then present an approach that allows verifiers to conduct cryptographic audits without having to plan for it prior to an election.

In the third part we examine how the methods in the first part can be extended to provide a level of privacy/distribution of trust similar to that of classical cryptographic voting protocols, while maintaining the (comparatively) intuitive optical-scan interface. To that end, we propose a novel paradigm for secure distributed document printing that allows optical-scan ballots to be printed in a way that still lets voters check their ballots have been counted, while keeping their voting preferences secret from election officials and everyone else.

Finally we outline how the results obtained in each of the three parts can be combined to create a cryptographically end-to-end verifiable voting system that simultaneously offers a conventional optical-scan ballot, ballot secrecy assured by a distribution of trust, and a simple, cryptographically austere set of audit procedures.

Acknowledgements

I would like to say a very special thanks to my co-advisors Urs Hengartner and Carlisle Adams. Both gentlemen are superb advisors and I feel enormous gratitude for the countless ways in which they supported my personal and professional development over the years.

I am honored to have had Rolf Haenni, Ian Goldberg, Alfred Menezes, and Doug Stinson examine this dissertation, and thank them for graciously lending their time and expertise to its improvement and completion.

Special thanks to Jeremy Clark, a dear friend and long-time research collaborator. I believe Jeremy has come to know the crypto voting literature in greater detail than anyone else in the field, so it is with great appreciation that I have been able to spend countless hours discussing this work with him.

Much of the work found in Part I was done with the Scantegrity team: Richard T. Carback, David Chaum, Jeremy Clark, Travis Mayberry, Stefan Popoveniuc, Ron Rivest, Emily Shen, Alan Sherman, Poorvi Vora, and Filip Zagórski. Working with this group was a tremendous experience. Thanks to David Chaum for encouraging me to pursue a dissertation on trustworthy voting. Very special thanks to Poorvi Vora for all her guidance and support.

I also wish thank the good people of Takoma Park, especially Director of Elections Anne Seargent and City Clerk Jesse Carpenter. Their remarkable vision has greatly advanced the cause of trustworthy voting.

Finally, a very heartfelt thanks to my family. In particular I owe a significant amount to my wife Anna. Despite the hardships and uncertainties of post-graduate life, she encouraged me to follow a dream. I remain eternally grateful for a leap of faith she made with me to change universities part way through. As in the words of Frost, it has made all the difference. Special thanks to my parents Chris and Sheran, and parents in-law Shirley and Ed, who went above and beyond to help us through that tumultuous time. I could not have completed this work without their love and support.

This research was supported in part by a Natural Sciences and Engineering Research Council of Canada (NSERC) Alexander Graham Bell Canada Graduate Scholarship.

Contents

List of Tables	xvii
List of Figures	xx
Preface	xxi
1 Introduction	1
1.1 Problem Statement	2
1.2 Thesis Statement	3
1.3 Contributions	4
1.3.1 Organization and Outline of Contributions	4
1.3.2 Contributions in Context	6
2 Preliminaries and Related Work	9
2.1 Preliminaries	9
2.1.1 End-to-end Correctness Proofs	10
2.1.2 Requirements for Ballot Secrecy	10
2.1.3 Privacy-preserving Receipts	11
2.1.4 Practical Requirements for Verifiable Elections	12
2.1.5 Limitations of End-to-end Verification	13
2.1.6 Document Security Primitives	14
2.1.7 Cryptographic Preliminaries	16
2.2 Related Work	18

2.2.1	Early Previous Work	18
2.2.2	Transitional Work: Voting Unassisted	21
2.2.3	Concurrent Work	22
2.2.4	Real-world Deployment	25
I	Interface Improvements for Voters and Election Officials	27
3	Scantegrity	29
3.1	Introductory Remarks	29
3.2	Voter Experience	30
3.3	The Switchboard	31
3.3.1	Auditing the Switchboard	32
3.4	System Architecture	35
3.5	Resolving Disputes	36
3.6	Implementation	38
3.7	Concluding Remarks	39
4	Scantegrity II	41
4.1	Introductory Remarks	41
4.2	Scantegrity II Procedures	43
4.2.1	The Vote Casting Procedure	43
4.2.2	Election Audit Procedures	47
4.2.3	Dispute Resolution Process	48
4.3	Cryptographic Proof of Tally	49
4.3.1	Ballot Definition	49
4.3.2	Roles	49
4.3.3	Functions	50
4.3.4	Trusted Computation Platform	53
4.3.5	Protocol	54

4.3.6	Correctness Proofs	61
4.4	Security Analysis	63
4.4.1	Assumptions	64
4.4.2	Manipulation Attacks	65
4.4.3	Identification Attacks	67
4.4.4	Disruption Attacks	68
4.5	Invisible Ink	69
4.5.1	Invisible Ink Threat Model	70
4.5.2	Invisible Ink Assumptions	71
4.5.3	Procedures For Printing With the Inks	72
4.6	Concluding Remarks	73
5	Scantegrity in Practice at Takoma Park, MD	75
5.1	Introductory Remarks	75
5.2	The Setting	76
5.2.1	About Takoma Park	77
5.2.2	Agreement with the City	77
5.2.3	Timeline	78
5.3	Scantegrity Overview	79
5.4	Implementation	83
5.4.1	Cryptographic Backend	84
5.5	The Election	89
5.5.1	Preparations Prior to Election Day	89
5.5.2	Election Day	92
5.5.3	After the Election	93
5.6	Surveys and Observations of Voter Experiences	96
5.7	Discussion and Lessons Learned	99
5.8	Concluding Remarks	100

II	Technical Simplifications for Election Auditors	103
6	Aperio	105
6.1	Introductory Remarks	105
6.2	Basic Paper Scheme	106
6.2.1	Ballot Format	106
6.2.2	Initial Setup	107
6.2.3	Print Audit Selections	110
6.2.4	Voting	110
6.2.5	Election Outcome	110
6.2.6	Decomitting	111
6.2.7	Receipt Audit	111
6.2.8	Tally Audit	112
6.2.9	Print Audit	112
6.3	Security Analysis	113
6.3.1	A Positive Assertion of Security	113
6.3.2	Prevented Attacks	114
6.4	Privacy	116
6.4.1	Prevented Attacks	117
6.5	Concluding Remarks	117
7	Eperio	119
7.1	Introductory Remarks	119
7.1.1	Motivation	120
7.2	The Eperio Protocol	121
7.2.1	Protocol Sketch	121
7.2.2	Entities	122
7.2.3	Functions	123
7.2.4	Lists and Tables	124
7.2.5	Protocol	126

7.2.6	Verification	129
7.3	Security	131
7.4	Practical Primitives	132
7.5	Implementation	134
7.6	Performance Comparison	137
7.7	Concluding Remarks	139
8	CommitCoin	141
8.1	Introductory Remarks	141
8.2	Preliminaries and Related Work	142
8.3	Commitments with Carbon Dating	144
8.3.1	Puzzle Properties	145
8.3.2	Limitations	146
8.4	Carbon Dating with Bitcoin	146
8.4.1	The CommitCoin Protocol	146
8.5	A (Simplified) Example of CommitCoin	148
8.6	Use with Scantegrity	149
8.7	Concluding Remarks	149
III	Paper-ballot Based Elections with Distributed Trust	151
9	Toward Oblivious Ballot Printing: A Two-party Approach	153
9.1	Introductory Remarks	153
9.1.1	Visual Crypto Preliminaries	154
9.2	Print an Arbitrary-length Secret Using a Dealer	154
9.3	Print a Randomly Selected Secret without a Dealer	157
9.4	Print a Permutation of a Set of Messages without a Dealer	160
9.5	Efficiently Print Text	162
9.6	Concluding Remarks	164

10 Scantegrity 3-D: Scantegrity with Distributed Trust	165
10.1 Introductory Remarks	165
10.2 Preliminaries	167
10.2.1 Physical Primitives	167
10.2.2 Cryptographic Primitives	167
10.2.3 Participants	168
10.3 The Basic System	169
10.3.1 Election Preparation	171
10.4 Verifying the election	175
10.4.1 Dispute Resolution	175
10.5 Improved System	177
10.5.1 Self-blanking Confirmation Codes	179
10.5.2 The Improved Ballot	180
10.5.3 Changes to the protocols	180
10.6 Security Analysis of the Improved System	182
10.6.1 Assumptions	183
10.6.2 Correctness	184
10.6.3 Voter Privacy (Sketch)	186
10.7 Discussion	187
10.7.1 Technical Challenges	187
10.7.2 Usability Questions	188
10.8 Concluding Remarks	188
11 Multi-party Oblivious Printing	191
11.1 Introductory Remarks	191
11.1.1 The Oblivious Printing Model	192
11.2 Visual Cryptography in this Setting	193
11.2.1 Fixed Pattern Visual Cryptography	193
11.3 Obliviously Printing an Encrypted Message	194

11.3.1	Translation Table	194
11.3.2	Setup	195
11.3.3	The Protocol	195
11.3.4	Obliviously Printing an Arbitrary Plaintext	197
11.4	Obliviously Printing a Randomized Message	197
11.4.1	Generating and Obliviously Printing a DSA/Elgamal Keypair	199
11.5	Mitigating Contrast Drop-off with AND-ing Inks	200
11.6	Example Applications	201
11.6.1	Electronic Voting	201
11.7	Security Analysis	203
11.8	Concluding Remarks	204
12	Putting it all Together	207
12.1	Introductory Remarks	207
12.2	Hash-only Verification	208
12.2.1	A Special Purpose Hash-based Commitment	208
12.3	Basics and Election Setup	210
12.3.1	Linkage Lists	211
12.3.2	Election Challenges and Audit	213
12.3.3	Verification in a Spreadsheet	214
12.4	Distributing Trust	215
12.5	Concluding Remarks	216
13	Discussion, Future Work, and Conclusion	217
13.1	Discussion	217
13.2	Future Work	219
13.3	Final Remarks	222
	References	222

Appendices	241
A Eperio Implementation Details	241
A.1 Eperio Audit Data Specification	241
A.2 Python Code for Eperio Verification	243
A.3 User Instructions for Manual/Spreadsheet Verification	244
B Carbon Dating the 2011 Takoma Park Pre-election Commitments	247

List of Tables

5.1	Results of the 2009 Takoma Park Election	94
7.1	Technical Requirements for Election Verification	137
7.2	Comparison of Election Verification Times	139
10.1	Scantegrity 3-D Notations	169
10.2	Diagram of Scantegrity 3-D Mark Transformations	175
10.3	Self-blanking VC Pixel	179
10.4	Printing a Self-blanking Invisible Ink Confirmation Code	181

List of Figures

1.1	Roadmap of Research in Relation to Previous Work	7
3.1	The Scantegrity Ballot	31
3.2	The Switchboard	33
3.3	Auditing the Printing of a Scantegrity Ballot	34
3.4	Scantegrity Receipt Check	35
3.5	Scantegrity Election Process Flow	37
3.6	Scantegrity Dispute Resolution Procedure	38
4.1	Scantegrity II Ballot	44
4.2	Example of the Scantegrity II master list P	53
4.3	Example Scantegrity II Switchboard	56
4.4	Topology of the Scantegrity II Switchboard	57
4.5	Example Switchboard Tables (Post-election)	59
4.6	Decoder Pen Activating Invisible Ink	70
4.7	Invisible Ink Printing Process	71
4.8	UV-Reactive Invisible Ink Camouflaging	73
5.1	Timeline of 2009 Takoma Park Election	78
5.2	Sample Ballot, Takoma Park 2009	81
5.3	Example Takoma Park/Scantegrity II Ballot and Verification Card	82
5.4	Scantegrity Election Workflow	83
5.5	Voter responses to Survey Questions 5, 9, 10, 13.	98

6.1	Aperio Ballot Assembly	107
6.2	Aperio Ballot Assembly (Exploded View)	108
6.3	Aperio Commitment Lists	109
6.4	Reconstituting the Receipt Audit Trail in Aperio	112
6.5	Reconstituting the Ballot Audit Trail in Aperio	113
7.1	Eperio Ballot and Table	124
7.2	Auditing Eperio Table Instances	130
7.3	Eperio Election Generation Wizard	135
9.1	Distributed printing	155
9.2	An Obviously Printed Document	157
9.3	Visual Crypto Up-sampling	163
10.1	The basic Scantegrity 3-D ballot.	170
10.2	Optical-scan oval with self-blanking confirmation code	180
12.1	A Basic HOVER Ballot and Audit Table	212

Preface

ON POLLS AND PROOFS

A voter once said: 'I think this election went awry.'

'Let me give you a proof,' came a cryptographer's reply.

Said the voter: 'Proof? What is that, and why should I care?'

'A proof is anything that convinces me,' said the cryptographer with a stare.

Now a sound one just might when it's right

So by chance, at first glance, our plan seems alright.

But proving you didn't cheat can be quite a feat

When the voter's path lay not in math... even if the proof's complete!

For the voter may wince when told to be convinced

By a proof made in zero-knowledge requiring a degree from college.

Indeed I concede: regarding perfection of elections

Cryptographic proofs are the worst form of crime.

Except for all those other proofs that have been tried from time to time...

for Anna

Chapter 1

Introduction

It's not the voting that's democracy;
it's the counting.

Tom Stoppard, *Jumpers*

Over the last decade we have witnessed a fundamental change in how elections are being conducted around the world: many voters now cast ballots through electronic means. As one particularly compelling example of the adoption of electronic voting technologies, consider the United States. In 2008, over 90% of jurisdictions (accounting for almost 99% of the population) used electronic systems for voting [Uni08].

However, software execution is not inherently observable, generating fundamental and widespread concern over the prominent use of such devices in the democratic process. Furthermore, beginning in 2004 with Kohno *et al.* [KSRW04], independent reviews have continued to uncover serious security vulnerabilities in widely-used commercial systems; vulnerabilities that allow, among other things, undetectable manipulation of election results when an adversary gains access to the system, even under reasonable threat scenarios. Recent studies have examined systems by ES&S [ACC⁺08], Premier/Diebold [CFH⁺07, BEH⁺08], Sequoia [BCE⁺07, AGH⁺09, CFK⁺09], and Hart-InterCivic [IRS⁺07, BEH⁺08].

In response, some have advocated a return to hand-counted paper ballots.¹ Tallying by hand is known to be an effective solution for relatively straightforward elections (e.g., at the Canadian Provincial and Federal levels), but the approach does not scale well in general. This can be a deal-breaker when considering complex tabulation rules such as the single-transferable vote (STV) counting method used in Australia and elsewhere. It also cannot accommodate the frequent, long and complex ballots common to many U.S.

¹See e.g., <http://blackboxvoting.org/>

jurisdictions. Consider, as an example, the logistical needs of a mid-to-large sized United States county such as Travis County, Texas. Servicing half a million registered voters across 210 precincts, in 2010 alone Travis County held 264 general-, special-, bond-, board-, runoff-, proposition-, and primary-elections, on 5 separate dates, for offices at the federal-, state-, county- city-, school/college-, municipal utility- and political party-levels of government.² Given this level of complexity, and despite an awareness of the potential risks, the Travis County County Clerk has indicated electronic voting is their only viable option.³ Many jurisdictions across the United States and elsewhere feel similarly, and the adoption of electronic voting is a trend we can expect to see continue into the foreseeable future.

And although hand-counted ballots offer a high degree of transparency to individuals physically present to observe the tally, it is often unrealistic for a single observer to witness the entire election from end to end. Typically, trust must be delegated to others when considering more than a single polling place (or in the event the ballots are recounted at a later date). The correctness of an election therefore often relies on the integrity of a trusted physical *chain-of-custody*. Of course even this limited security model is mooted by the introduction of electronic components.

Cryptographic Election Verification. We seek a solution that provides compelling evidence, to anyone, at any time after the election, that all precincts reported accurate results. We seek a solution that certifies the correctness of the election results, not of the voting equipment. Ultimately we seek an informational solution that can provide two seemingly incongruous properties: proof that an election was counted correctly, while simultaneously protecting ballot secrecy. Our solution rests in *cryptographic* election verification.

Cryptographically end-to-end (E2E) verifiable voting allows election outcomes to be independently and universally verified by members of the public. It provides voters with a special receipt of their cast ballot—one that allows them to verify their vote was included in the outcome, but does not reveal to anyone how they voted. The ability to check this receipt, combined with a special cryptographic proof of correctness establishes a high degree of confidence in the correctness of the election outcome.

1.1 Problem Statement

Over two decades after the seminal groundwork was laid by Chaum [Cha81], Benaloh [Ben87] and others, cryptographic election verification has been used in real elections with

²http://www.co.travis.tx.us/county_clerk/election/

³Dana Debeauvoir. Keynote Address. EVT/WOTE, 2011.

binding results (cf. [ECCP07a, AMPQ09, CCC⁺10]). Yet despite these advances in real-world practicality, the use of cryptography in elections has remained a point of contention. A consensus is emerging among researchers and election officials that conceptual and procedural simplification will be central to the future of cryptographic election verification.⁴ This trend is reflected in a 2009 German Federal Constitutional Court decision which states that if verification of a tally requires “specialist knowledge,” it is unconstitutional [Fed09].

As a minimum requirement of an election system, any proposed solution must not unduly burden users in the completion of their primary tasks, i.e., casting a ballot, tabulating votes, etc. Ideally, the procedures for cryptographically verifying an election would also be technically and conceptually simple so as to facilitate the greatest adoption and participation in the audits. As a trivial solution, procedural and conceptual simplifications can usually be made at the expense of security properties (and vice versa). The result has been a tension that has arisen between cryptographers and election officials in the design of trustworthy voting systems.

1.2 Thesis Statement

As we have attempted to briefly argue in the introduction, the design of electronic voting technology certainly appears to exhibit a tension between security and real-world practicality. Whereas the electronic voting equipment industry seems to be producing practical systems at the expense of security, so too does academia seem to be proposing secure protocols at the expense of practicality.

An important question in the design of electronic voting technology, therefore, is whether this seemingly zero-sum relationship between properties is a fundamental limitation of the design space, or simply a product of overly narrow design goals.

Thesis Statement. *With regard to the design of electronic voting technology, we claim security, privacy, and trustworthiness does not exist in a zero-sum relationship with real-world practicality. Indeed designers of electronic voting equipment can (and should) seek to offer both properties in conjunction.*

Objectives and Scope. Our objective is the design of a voting system that balances the real-world needs of voters, election officials, and auditors with the privacy and integrity properties of cryptographic end-to-end verifiable voting systems. In particular, our goal

⁴See for example talks by Epstein, Gilbert, Jones, Sergeant, Volkamer, and Wallach at the NIST Workshop on End-to-End Voting Systems, Washington D.C., 2009. <http://csrc.nist.gov/groups/ST/e2evoting/>

is the design of a voting system that combines the real-world practicality of a conventional *paper optical-scan* voting system with similar security properties as those offered by more theoretically minded proposals for *cryptographic* election verification. The proposed solution will seek to meet three objectives simultaneously:

- **Objective I: Improve real-world practicality for voters and election officials.** Design an end-to-end verifiable voting system with a familiar paper optical-scan voting interface—one that plausibly could be voted on and counted without requiring specialized knowledge, devices, or training,
- **Objective II: Improve real-world practicality for election verifiers.** Provide the public with a simplified cryptographic proof—one that could plausibly be verified without specialized software and with less emphasis on specialist knowledge,
- **Objective III: Distribute trust in paper ballot elections.** Achieve objectives I and II without relying on trusted entities/devices to protect ballot secrecy.

1.3 Contributions

In this section we outline the contributions of this dissertation and describe how they fit in with the thesis statement.

1.3.1 Organization and Outline of Contributions

This document contains excerpts from 10 peer-reviewed publications made in support of the thesis statement; each is organized into its own chapter. A list of contributions are presented at the beginning of each chapter. The chapters are organized as follows:

- **Preliminaries and related work**, presented in Chapter 2, describes core requirements and desired properties of cryptographically verifiable voting, and summarizes related work—both previous and concurrent with this research,
- **Interface Improvements for Voters and Election Officials:** Part I presents results for making E2E more practical for voters and election officials:
 - **Scantegrity**, a cryptographically verifiable add-on to a conventional optical-scan voting system [CEC⁺08], presented in Chapter 3,

- **Scantegrity II**, an improvement to the original proposal simplifying the dispute resolution process through the use of invisible ink confirmation codes [CCC⁺09], presented in Chapter 4,
- **Real-world election** including findings from a usability study [SCC⁺10] and the experiences implementing and running Scantegrity II in a municipal election [CCC⁺10], presented in Chapter 5.
- **Technical simplifications for election auditors:** Part II presents results for making E2E more practical for election auditors/verifiers:
 - **Aperio**, an E2E verifiable voting system based on tamper-evident documents conceived as a means of educating the general public about the end-to-end properties without requiring knowledge of cryptography [ECA08], presented in Chapter 6,
 - **Eperio**, an electronic (i.e., cryptographic) version of Aperio whose simplified back-end permits cryptographically verifiable election audits to be performed with minimal requirements for custom software (such performing verification in a desktop spreadsheet) [ECHA10], presented in Chapter 7,
 - **CommitCoin**, a scheme allowing election auditors to meaningfully verify a commitment-based election proof at any time after the election by presenting a proof-of-work based argument that a commitment was made sufficiently far in the past [CE12], presented in Chapter 8.
- **Paper-ballot elections with distributed trust:** Part III presents results for improving the privacy of the techniques presented in Part I through the distribution of trust in both the cryptographic back end, and the paper-ballot front end. It includes:
 - **Toward oblivious printing**, a two-party method for printing a randomized, human-readable secret [ECHA09], presented in Chapter 9,
 - **Scantegrity with distributed trust**, a Scantegrity variant with two-party distributed trust and ballots printed using the above technique [EHH11], presented in Chapter 10,
 - **Multi-party oblivious printing**, a generalization of [ECHA09] allowing the contents of an encryption to be printed in human-readable form, but for which none of the printers learn the result [EH12b], presented in Chapter 11.
- **HOVER**, presented in Chapter 12, presents an E2E verifiable voting system outlining how the results obtained in each of the three parts can be combined to create a cryptographically end-to-end verifiable voting system that simultaneously offers a

conventional optical-scan ballot, ballot secrecy assured by a distribution of trust, and a simple, cryptographically austere set of audit procedures.

- **Concluding remarks** are given in Chapter 13, summarizing how the results presented in this dissertation met the objectives set out in the thesis statement.

1.3.2 Contributions in Context

Our review of the literature begins in Section 2.2; it would bring context to the discussion of our contributions, however, to place them in terms relative to previous work. As a starting point, we broadly define previous work to mean cryptographically verifiable voting schemes with fully electronic interfaces, multi-party distribution of trust, and those that employ cryptographic proof techniques requiring extensive expert knowledge. One such example is the seminal voting scheme due to Cramer et al. [CGS97].

For illustrative purposes, we provide a high-level roadmap of our contributions relative to previous work in Figure 1.1. We proceed by meeting each thesis objective individually, and conclude with a sketch for combining the results into a single system.

These contributions and their relationship with previous work are:

- **Contributions toward objective I:** The work presented in Part I focuses on simplifying the ballot casting and counting procedures for voters and election officials through the use of paper optical-scan ballots. Printing paper ballots, however, requires a trusted entity, offering weaker protections of ballot secrecy. As depicted in Figure 1.1, this offers improved real-world practicality at the expense of decreased protections to ballot secrecy, relative to previous work,
- **Contributions toward objective II:** The work presented in Part II focuses on simplifying the protocols presented in Part I to reduce the conceptual and technical requirements of election verifiers. As depicted in Figure 1.1, this improves real-world practicality relative to the results of Part I.
- **Contributions toward objective III:** The work presented in Part III focuses on mitigating the privacy trade-off incurred by Part I through novel techniques for printing paper ballots in a distributed setting. As depicted in Figure 1.1, this improves protections to ballot secrecy relative to the results of Part I.
- **Final Result.** The final result of this dissertation, presented in Chapter 12, combines the properties achieved in Parts I, II, and III. As depicted in Figure 1.1, the resulting system offers distribution of trust that is comparable to previous work, while offering greatly improved real-world practicality to voters, election officials and verifiers.

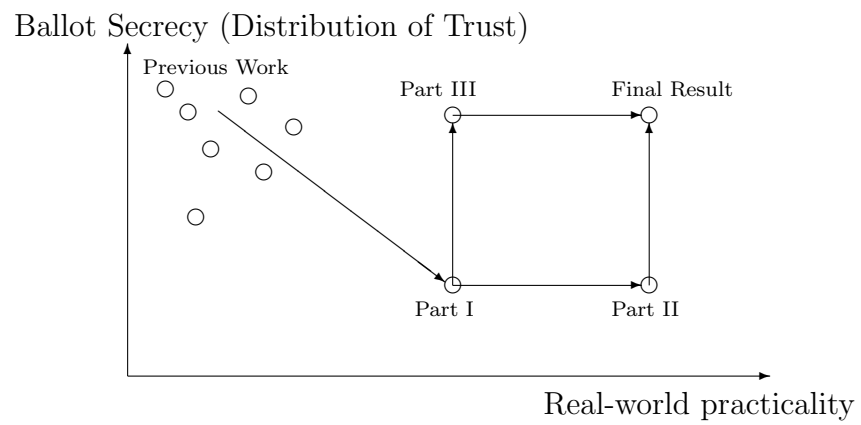


Figure 1.1: Roadmap depicting the direction of this research in relation to previous work. The result is a verifiable voting system that improves real-world practicality for voters, election officials, and verifiers while offering a distribution of trust comparable to previous work.

Chapter 2

Preliminaries and Related Work

In the cryptographic literature, electronic voting protocols are known as the prime examples of secure multi-party computations.

Ronald Cramer, Rosario Gennaro,
Berry Schoenmakers [CGS97]

2.1 Preliminaries

The ultimate goal of this research, and that of trustworthy voting in general, is to provide compelling and universal evidence (i.e., proof) that a given election outcome is correct. By correct, we mean that the given election result was counted as the electorate collectively intended. Proving that an election was *counted-as-intended* constitutes proving the correctness of all aspects of an election that transpire between when voters cast their ballots and when a result is declared. An election offering such a proof is said to be *end-to-end* (E2E) verifiable. An end-to-end verifiable election typically involves the following entities:

- A set of voters with the authority to cast a ballot in an election. Voters typically are issued (or themselves construct) a privacy-preserving “receipt” of their cast ballot that they can use as part of the wider election audit,
- A set of election officials/trustees who organize and run the election, report results, and produce a universal and publically verifiable (usually cryptographic) proof of election correctness,

- A set of verifiers/auditors whose job is to verify the correctness of the proof issued by the trustees.

2.1.1 End-to-end Correctness Proofs

Proving that an election was counted-as-intended typically involves the trustees simultaneously proving three (composing) properties to voters:

- **Cast-as-Intended:** At a high level, a voter votes by selecting their preferred outcome from a set of possible outcomes and then declaring this preference to the election trustees. The trustees issue a proof that establishes that they received the voter’s preference as it was declared (i.e., cast),
- **Collected-as-Cast:** Cast preferences are stored until all preferences have been received. When the election is complete, a collection of preferences will be aggregated according to a preference aggregation rule. The trustees issue a proof that this collection of preferences to be aggregated is equivalent to the set of all cast preferences,
- **Counted-as-Collected:** Finally, the trustees must declare an election outcome. The trustees issue a proof that establishes that the declared outcome is equivalent to the result of applying the preference aggregation rule to the collection of preferences.

Verification of these proofs typically occurs at two levels: **voter verification** in which voters individually check that their preferences are correctly reflected in the overall proof, and **universal verification** in which anyone can verify that the overall proof is consistent with the declared election result.

Soundness. The proposals discussed throughout this document implement the above correctness proofs through a variety of approaches. In contrast to conventional techniques (e.g., chain-of-custody) the soundness of these proofs are built on security assumptions other than simply trust. Soundness is typically premised around cryptographic and statistical assumptions, although in the case of Aperio (see Section 6), physical tamper-evidence is also considered.

2.1.2 Requirements for Ballot Secrecy

There has been some debate in the literature as to whether ballot secrecy is a fundamental requirement of end-to-end verification, or not. Popoveniuc et al. [PKRV10] argue, as does Jones in a position paper [Jon09], that ballot secrecy should be regarded as a separate and orthogonal property. Despite this, one of the first uses of the term “end-to-end” was made

by the United States Election Assistance Commission [Uni05] in which they define an “end to end cryptographic independent verification system” as one which “issues a paper receipt to the voter that contains information that permits the voter to verify that the choices were recorded correctly” but “does not permit the voter to reveal his or her selections.”

For the purposes of this document, and consistent with our previous work, we define end-to-end verification in the context of *secret-ballot* elections. We outline ballot secrecy in terms of protection against an adversary attempting to learn voting preferences in the following scenarios:

- **Passive secrecy:** observing a public transcript of the protocol is not sufficient for an adversary to recover a voter’s cast preference,
- **Distribution of trust:** observing the private inputs of some number of trustees (below a critical threshold) is not sufficient for an adversary to recover a voter’s cast preference,
- **Receipt-freeness:** a voter is unable to create a receipt/vote obfuscation that reveals to an adversary information about their cast preference,
- **Coercion-resistance:** following a secure registration protocol between a voter and trustees made prior to an election, an adversary instructing a voter to vote in a certain way is unable to distinguish between compliance and non-compliance, despite being able to observe the voter’s private inputs during voting. See Juels et al. [JCJ05] for a formalization.

All the proposals discussed below achieve passive secrecy (unless noted). The presence of distribution of trust and receipt-freeness vary across proposals. Coercion resistance was proposed mainly in the context of Internet voting and is mostly out of scope for work on in-person/polling place voting.

2.1.3 Privacy-preserving Receipts

The receipt paradigm is an integral part of the end-to-end proof methodology. During the voting process a voter is given the option to receive (or construct) a “receipt” of their vote. The receipt allows the voter to verify that their preference was collected-as-cast, while simultaneously enforcing ballot secrecy (i.e., the receipt is receipt-free).

Clarification on the Dual Use of “Receipt.” The term “receipt” often appears in two different (though related) contexts in the literature. Our usage in this dissertation is as follows:

- **Receipt-free** (*adjective*): A guarantee of ballot secrecy as defined above,
- **Receipt** (*noun*): An *obfuscation* of a voter’s preference, presented to the voter when they cast their ballot, allowing them to later verify their preference was collected-as-cast by the trustees.

Vote Obfuscation. There are several approaches to obfuscating a voter’s preference. Examples from the literature include:

- **Encrypting a preference:** A voter encrypts their preference. This encryption is cast and a copy is retained as a receipt, e.g., [Adi08, JCJ05, CCM08],
- **Selecting a preference from a randomly permuted a list of choices:** A list of choices is presented to the voter in a randomized order (relative to some canonical ordering). The position of the voter’s preferred choice within this randomized list is cast and a copy is retained as a receipt, e.g., [CRS05, PH06, AR06, ECA08],
- **Substituting a preference for a randomized code:** A list of choices is presented to the voter. Each choice is associated with a randomized code. The code associated with the voter’s preferred choice is cast and a copy is retained as a receipt, e.g., [BMR07, CEC⁺08, CCC⁺08],
- **Secret sharing a preference:** A voter’s cast preference is secret-shared in some fashion. Some number of shares sufficient to recover the voter’s preference are cast. The voter retains a copy of some number of shares (insufficient to recover the preference) as a receipt, e.g., [Cha04, RS07],
- **Floating receipts:** A voter casts their preference unobfuscated. The voter receives a copy of the unobfuscated voting preference of another (random and anonymous) voter as a receipt, e.g., [Cus01, RS07].¹

2.1.4 Practical Requirements for Verifiable Elections

Several important considerations must also be made in the design of end-to-end verifiable voting system intended for a real-world setting:

- **Procedural simplicity:**
 - **Primary election tasks:** Users should be able to accurately complete their primary tasks pertaining to the election,² i.e., voting, election administration, etc,

¹We include such systems, noting they have been criticized for not fully achieving the E2E properties [ECA08].

²This is also true of conventional voting systems.

- **Secondary verification tasks:** Users should be able to complete their secondary tasks pertaining to election verification, i.e., receipt creation, receipt checking, anti-coercion strategy, proof generation, proof verification, etc.
- **Conceptual simplicity:** The details of the cryptographic proof should be conceptually accessible to the widest possible audience within the constraints of the required security properties,
- **Psychological acceptability:**
 - **Mental model:** The voting interface should not plausibly be in conflict with a voter’s mental model, for example, by leading the voter to (erroneously) believe they are being instructed to destroy their ballot (cf. [ECCP07a]), or are being required to cast a preference for a disagreeable candidate (cf. [JJB06]),
 - **Modularity of cryptographic components:** Cryptographic end-to-end verification of an election should not fundamentally interfere with other existing verification methods, e.g., a voter-verified paper audit trail (VVPAT).
- **Legal requirements:** Any proposed solution must comply with local election laws. Legal compliance can be a design challenge as many jurisdictions require, e.g., fixed-order candidate lists and unobfuscated paper-audit trails.

2.1.5 Limitations of End-to-end Verification

Cryptographic end-to-end election verification is by no means a silver-bullet solution to protecting the democratic process. We provide a brief (non-exhaustive) list of examples of aspects of the democratic process not protected by the end-to-end model:

- **Voter suppression:** impediments to voting (some legal, some illegal) including:
 - Deleting voter lists to disrupt registration,
 - Disenfranchising certain voters through polling place taxes and literacy tests,
 - Under-resourcing certain polling places to cause overly long line-ups,
 - Partisan challenges to the eligibility of certain voters, for example, through the use of caging lists,
 - Voter intimidation,
 - Dis-/mis-information propagated on election day, for example, designed to cause voters to go to the wrong polling place, or perhaps to mark their ballot incorrectly,

- **Partisan gerrymandering:** redrawing of electoral districts to advantage a particular candidate/party by diluting (or concentrating) the influence of a particular demographic on the election outcome,
- **Registration fraud:** attacks stemming from improper registration including ballot stuffing and voter impersonation,
- **Denial of service:** attacks ranging from simple power outages at the polling place, overloading web-servers with traffic (as in the case of internet voting), or submitting numerous phony votes (so-called “board flooding”) in coercion resistant internet voting schemes to overload the computational capacity of election officials attempting to cryptographically validate submitted ballots.
- **Side-channel attacks:** vote buying/selling made possible by through side-channel attacks at the polling place such as the (illegal) use of cell-phone cameras in the voting booth to provide evidence of one’s vote.

See Popoveniuc et al. for further detail [PKRV10]. In many cases, and unlike the kinds of fraud cryptographic verification seeks to detect, the occurrence of one of the above attacks leaves sufficient evidence to be protected against by legal or policy based (as opposed to technical) measures. Finally, it is worth noting that even under ideal scenarios in which all avenues of electoral fraud are discounted, the impossibility theorem due to Arrow [Arr50] proves no method exists for aggregating voter preferences into an election result that is guaranteed to be fair under a particular set of reasonable requirements.

2.1.6 Document Security Primitives

Invisible Ink. Invisible ink refers to a chemically reactive ink with the following properties:

- In its initial (unactivated) state, it is clear/unpigmented,
- When combined with a developing agent, an irreversible chemical reaction occurs (activation) causing the ink to darken/pigment.

Simple invisible inks can be derived from common acidic household substances, such as lemon juice or vinegar. A solution of such a substance diluted in water, can be used to write a secret message onto a sheet of paper. This clear solution, once dried, can be developed (*i.e.*, activated) by exposing the paper to a heat source thereby revealing the secret message.

The term “invisible ink,” however, is somewhat of a misnomer. Messages printed in invisible ink are not fully invisible, since any liquid changes the reflective characteristics of the paper it is printed on. For this reason we employ a second non-reactive “dummy” with the following properties:

- Its characteristics (pigmentation, reflectivity, etc) are visually indistinguishable from unactivated reactive ink,
- Contact with the developer ink will produce negligible pigmentation change, making it easily visually distinguishable from activated reactive ink.

Messages initially printed using reactive ink are followed with an application of non-reactive ink in the negative space (i.e., the space around and between the message), thereby creating a region that is uniformly reflective, thus concealing the message. Alternatively, as will be explored in future chapters, the dummy ink could also be a chemically reactive ink with an anti-catalyst to slow its development. Printing in this way would allow a message to be visible for only a brief period of time after activation. The ideal security properties of invisible ink printing are:

- **Invisibility:** Messages printed in invisible ink should be unreadable prior to activation,
- **Activation-evident:** Activated ink should be irreversible and plainly evident.

Work presented in Chapters 4 and 5 undertakes development of invisible inks for verifiable optical-scan election ballots. In this setting, the inks are specifically designed to be both amenable for use with commercial-off-the-shelf ink-jet printers, as well as activated by a developer chemical placed into a special-purpose pen (as opposed to a heat source). Technical improvements to the indistinguishability of invisible ink, especially under black light, employ randomized overprinting of variably ultraviolet-reactive (but non-activating) inks to effectively camouflage the message when viewed under an assortment of wavelengths of light.

Document Authentication. We rely on document authenticity throughout Part III to prevent a class of attacks potentially allowing an adversary to undetectably learn secret information through the execution of an oblivious printing protocol. During such an execution, each party (i.e., printer) prints their share of a secret on the document in invisible ink. Document authentication prevents a corrupted printer from activating the invisible ink applied by previous printers to learn the shares of other printers, and then reprinting the document in an effort to escape detection.

Potential techniques for fingerprinting physical paper documents include paper color, paper texture, and ink splatter. It was shown by Buchanan et al. [BCJ⁺05] that fiber patterns can be used to uniquely identify paper documents. Recent work due to Clarkson et al. proposes a practical scheme based on fuzzy feature extraction from the three dimensional shape of the paper [CWF⁺09]. This fingerprinting scheme can be implemented using a commodity scanner and is robust against additional ink being printed on the paper, as well as light mishandling of the document. Sharma et al. [SSB11] implement a paper fingerprinting scheme based on texture speckles using a USB microscope costing under \$100.

Document fingerprinting is outside of our scope. For the sake of this dissertation, we assume that there exists an practical, efficient, and accurate paper fingerprint scheme through which printers can determine whether the same physical sheet of paper was present at each stage of the protocol’s execution.

2.1.7 Cryptographic Preliminaries

We briefly outline some of the main cryptographic primitives used by our systems. We note that these primitives are standard across the cryptographic voting literature.

Visual Cryptography. A visual cryptography scheme (VCS) is a visual secret sharing scheme in which a (secret) message or graphical image is split into a number of shares. Individually the shares reveal no information about the message. However when any permissible combination of shares are combined, the message becomes visually perceptible. In its classical manifestation, visual crypto shares are printed on transparent plastic sheets using conventional inks. The sheets are stacked (i.e., overlaid) to recover the message. Creation of the shares is typically undertaken by a “dealer” who knows the message and enforces its secrecy across any non-permissible combination of shares.

An early example of visual secret sharing is due to Kafri and Keren [KK87] (what they call “random grids”), although Naor and Shamir [NS94] are generally credited with the paradigm in the security literature. The latter outline a collection of visual crypto schemes for which the shares of some threshold $k > 2$ out of n printers are necessary to recover the image and is denoted as (k, n) -VCS. Ateniese et al. [ABSS96] generalize this notion to access structures for which the message is recoverable under arbitrarily defined subsets of participants. A survey of a number of variations of visual cryptography is presented in [Yan11]. Other pertinent references are the perceptual effects of misaligned shares [LWL09], the use of seven-segment displays with VC [Bor07], and the application of VC to electronic voting [Cha04].

Homomorphic Encryption. Let $\langle \text{DKG}, \text{Enc}, \text{DDec} \rangle$ be a *distributed public-key encryption* scheme. Without loss of generality, DKG generates two private key shares x_1 and x_2 for parties \mathcal{P}_1 and \mathcal{P}_2 respectively and a joint public key Y . Encryption $\llbracket m \rrbracket = \text{Enc}_Y(m, r)$ is semantically secure and homomorphic in at least one operation. Decryption $m = \text{DDec}_{(x_1, x_2)}(\llbracket m \rrbracket)$ requires both key shares. Specifically we will make use of exponential Elgamal [CGS97] with distributed decryption [Ped91]. For simplicity we will omit the public key when implied.

Mixnets. Mixnets have long been a fixture in cryptographic voting. We will make frequent use of reencryption mixnets (cf. [PIK93]) to create our proofs. *Rerandomization* of a ciphertext c (often referred to in the literature as reencryption—especially in the context of mixnets) is accomplished by computing $c' = \text{ReRand}(c, r) = c \cdot \text{Enc}(0, r)$.³ By rerandomizing and shuffling a batch of ciphertexts we implement a simple *reencryption mixnet*, Mix . When applying Mix to a matrix of ciphertexts, we describe mixing as occurring on *tuples* of ciphertexts grouped by columns and shuffled by rows.

Commitments. Briefly, a cryptographic commitment scheme $\text{Comm}(m, r)$ takes message m and randomness r and produces commitment c . $\text{Open}(c, m, r)$ takes the commitment and asserted message m and randomness r and returns *accept* iff $\text{Comm}(m, r) = c$. A commitment is said to be *binding* if it is hard to find any $\{m, m', r, r'\}$ where $m \neq m'$ such that $\text{Open}(\text{Comm}(m, r), m', r')$ accepts. A commitment is said to be *hiding* if it is hard to find any (m, r) given c such that $\text{Open}(c, m, r)$ accepts. As a stronger requirement of hiding, it should also be hard to extract partial information about m given c .

Fair Cut-and-choose Challenges. A staple part of the cut-and-choose correctness proofs contained throughout this document is the ability to generate random challenges in a way that is fair. Loosely speaking, *fairness*, requires that no one is able to predict, or reliably influence the output with non-negligible advantage. Furthermore, the fairness of the method should be somehow observable to voters. Both the heuristic due to Fiat and Shamir [FS86], and the notion of a *random beacon* (cf. [Rab83, CH10]) are possibilities. Clark et al. [CEA07b] have suggested the use of financial data (e.g., stock market closing prices) as a means of generating election audit challenges, which may be considered fair and observable on the assumption that reliable and fine-grained market manipulation is infeasible. Clark and Hengartner later showed [CH10] that stock closing prices provide sufficient entropy for this purpose.

³Replace 0 with the *identity* element for other groups.

2.2 Related Work

Research into cryptographically verifiable voting entered its 30th year in 2011. The past three decades have seen considerable advancement in this field. As an important application of cryptography, advances in voting have coincided with a number of broader advances, including mixnets, additively homomorphic encryption, distributed decryption and others. This history might roughly be divided across three eras:

- **Early work:** seminal early work beginning in the early 1980's to the mid 2000's,
- **Transitional work:** a period marking a move toward real-world practicality beginning in the mid 2000's, including the origins of this research,
- **Concurrent work:** recent work done concurrently with this research (i.e., 2008–2012).

2.2.1 Early Previous Work

Mixnets and the First Crypto Voting Scheme. The first cryptographic voting protocol was proposed in 1981 by Chaum in his seminal paper on mixnets [Cha81]. The scheme combines the use of (non-verifiable) decryption mixnets with digital signatures. Voters are responsible for creating their own signing key pair. Voters submit their verification key through the mixnet (operated by the trustees) allowing the trustees to create a roster of authorized (but anonymized) signing keys. In the second phase, voters send a signed vote through the mixnet. Trustees ensure that each received vote corresponds to a valid signature in the roster. They then tally all qualified votes. Voters can individually check their vote appears in the list of qualified votes, and anyone can repeat the tally (since the qualified votes are in cleartext). The scheme offers passive secrecy and distributed trust but is clearly not receipt-free: a voter can betray their cast preference by revealing their signing key.

Chaum later proposed a voting scheme based on DC-Nets [Cha88] offering unconditional passive secrecy. Bos later refined this approach [Bos92] using Pedersen commitments [Ped92].

Homomorphic Tallying. In addition to mixnet-based voting, one other approach has become prevalent: homomorphic tallying. As the name suggests, and in contrast to mixnet-based voting, homomorphic tallies are computed under encryption. In its basic form a voter encrypts their preference using an additively homomorphic encryption scheme, casting the ciphertext. Trustees can compute a tally by decrypting the product of all cast ciphertexts.

The primary technical challenges at the time were twofold: one was to create an additively homomorphic encryption scheme. The other was to develop a means for a voter to prove that the ciphertext they cast contained a valid preference. Being an additive scheme, it is fundamental to correctness that a voter not be able to over-vote (or subtract votes!).

Benaloh introduced this direction via a series of publications [BF85, Ben86a, Ben86b]. His additively homomorphic encryption scheme was also an important step in its own right: it improved upon the encryption scheme of Goldwasser and Micali [GM84] by allowing the homomorphic addition of much larger numbers. Although decryption in Benaloh’s scheme was still a hard problem in general, it was feasible to search the set of possible outcomes of any plausibly sized election. His scheme also paved the way for Paillier’s additive scheme [Pai99], which allowed generalized decryption.

Techniques for distributed decryption of Benaloh’s cipher were not available at the time and so follow-up work included various approaches for achieving distribution of trust by having voters essentially act as dealers in a secret sharing scheme, breaking their votes into shares [BY86, Ben87]. Sako and Kilian [SK94] and Cramer et al. [CFSY96] later presented several optimizations. The latter introduced a number of useful proof techniques still widely used today.

Receipt-freeness. Benaloh and Tuinstra eventually introduced the notion of receipt-freeness [BT94], a property that enforces ballot secrecy even if the voter intends to disclose their preferences (e.g., for the purposes of vote-selling). In contrast to the previous homomorphic proposals, the voter does not create the encryption of their own preference. Instead, the trustees create the encryptions of all potential choices, leaving the voter to select and cast the ciphertext corresponding to their preference.

Of course, the trustees must be able to prove to the voter that the chosen ciphertext correctly encrypts the voter’s preference, but without giving the voter information to prove this fact to others. At a high level, this is accomplished by a cut-and-choose protocol. In its most basic form the trustees transmit n ciphertexts to the voter, each randomly encrypting either a “yes” or a “no” vote for a particular candidate. For each such ciphertext, the trustees assert the corresponding plaintext. Based on these assertions, the voter chooses one of the ciphertexts corresponding to their preference, and challenges the trustees to verifiably decrypt the others. The voter checks the resulting plaintexts match the corresponding assertions. When satisfied, the voter’s chosen ciphertext is cast. Since the voter did not learn the random factor used to encrypt their preference, the voter is unable to prove its contents to a third party despite being confident of the plaintext. Hirt and Sako [HS00] later demonstrated an attack on receipt-freeness of Benaloh and Tuinstra’s scheme.

Universally Verifiable Mixnets. Compared to the advances in homomorphic tallying, little had been published on mixnet voting schemes in the decade since Chaum’s original proposal. Park et al. [PIK93] eventually proposed the reencryption mixnet as an alternative to DC-Nets used in Chaum’s second scheme. Sako and Kilian used this to present the first universally verifiable mix network [SK95].

Recall that in Chaum’s original voting scheme, the mixnet was not verifiable. This meant a voter had to rely on some additional mechanism (digital signatures in this case) to be able to receive assurance their vote was correctly counted. Recall that by contrast, Benaloh’s solution to receipt-freeness involved the trustees preparing the ballots on the voter’s behalf instead. So given that a voter could not individually verify the mixnet, the technical challenge at the time involved making the mixnet verifiable to anyone (i.e., universally verifiable). Their solution rests on a cut-and-choose protocol similar to a proof of graph isomorphism. The resulting voting system is receipt-free⁴ and has distributed trust, but carries a considerable computational cost relative to homomorphic-based proposals.

Distribution of Trust in Homomorphic Schemes. Leveraging advances in homomorphic encryption schemes occurring since Benaloh’s early work (i.e., ElGamal), Cramer et al [CGS97] presented a proposal for a voting scheme that offered not just distributed trust, but *threshold* distributed trust. Their proof techniques were very efficient compared to earlier proposals and form the basis of many systems even today. It still leaves the voter to prove the validity of their own vote and is not receipt-free as a consequence. Hirt and Sako later extended this work by adding receipt-freeness and introducing efficient techniques for combining multiple candidate counters into a single ciphertext [HS00].

Coercion Resistance. Juels et al. [JCJ05] defined the notion of coercion-resistance as a privacy protection to voters casting ballots in a non-private environment (e.g., the Internet). The intention here is to describe a scenario in which an adversary can corrupt a voter—even as they are casting their ballot. In addition to receipt-freeness, they define coercion-resistance as also protecting against the following attacks:

- **Randomization:** An adversary shall not be able to receive proof as to whether a voter complied with the adversary’s directive to cast a random preference,
- **Forced abstention:** An adversary shall not be able to receive proof as to whether a voter complied with the adversary’s directive to abstain from casting a ballot,⁵

⁴They assume trustees transmit the proof of ballot correctness to the voter over a deniable channel.

⁵This requirement may conflict with the laws of certain jurisdictions, for example, in Australia, where voting is mandatory.

- **Simulation:** No adversary shall be able to undetectably cast a preference on behalf of a voter.⁶

The mechanism commonly employed to achieve coercion-resistance is to require voters to perform a private registration step prior to the election in which the voter receives a credential. The technical challenge is twofold: one is to make real and fake credentials indistinguishable from a coercer’s perspective, so they will not know when a real or fake vote is being cast. The other challenge is to make real and fake credentials distinguishable to the trustees so that fake ones can be restrained from inclusion in the tally.

2.2.2 Transitional Work: Voting Unassisted

Beginning in the mid-2000’s voting research entered a new phase. Until this point in the literature a voter was essentially modeled either as being an interactive Turing-machine, or as accessing a trusted computational assistant. Beginning with a proposal in 2004 by Chaum [Cha04], this new direction set out to develop means for human voters to be able to obfuscate their vote without requiring a computational assistant. Chaum’s scheme involved a voter interacting with a direct-recording electronic (DRE) touchscreen interface. The DRE would print a physical ballot showing the voter’s preference in human-readable form. The ballot itself consisted of two visual crypto shares (cf. [NS94]) which were separated. One layer was chosen randomly to be destroyed; the other was optically scanned, and the voter retained a copy as a receipt.

Later, Ryan et al. proposed Prêt à Voter [CRS05], the first system to offer a pre-printed optical-scan ballot. A Prêt à Voter ballot paper consists of two halves separated by a perforation: one half contains a list of candidates presented in an independent random order. The other half contains a location for a voter to mark a preference. When marking was complete, the voter detached the candidate list and destroyed it. The remaining portion was optically scanned, and the voter retained a copy as a receipt. Since the candidate list order was randomized, the vote can not be determined from the position of the voter’s mark alone.

With regard to receipt-freeness, these proposals take the physical analog to the receipt-free systems of the 90’s: a set of election trustees create the obfuscation on the voter’s behalf.

Punchscan. Chaum’s 2004 scheme proved challenging to deploy in practice on account of the double-layer visual-crypto ballot. He later proposed Punchscan as a (slightly) more practical, and purely optical-scan follow-up [PH06, FCS06].

⁶This is a non-trivial threat in remote/Internet voting. For in-person voting schemes, this is typically handled by conventional identification techniques (e.g., photo ID) and is out of the scope of this work.

Punchscan sacrifices distributed trust for simplified cryptographic proofs, but curiously offered a considerably less usable ballot relative to that used by Prêt à Voter. Similar to Chaum’s 2004 scheme, the ballot consisted of two stacked paper sheets. The top sheet consists of randomized codes beside a fixed-order candidate list. The bottom sheet contains codes in a different (i.e., independent) random ordering. Holes punched in the top sheet allow the codes on the bottom sheet to show through. Using a bingo dauber, the voter would mark the hole containing the same code as the one appearing beside their preferred candidate. Similar to the 2004 scheme, one sheet would be destroyed, and the other scanned and retained as a receipt.

We implemented Punchscan and deployed it in a graduate student election at the University of Ottawa [ECCP07a]. We later entered the implementation into VoComp, a voting system design competition in 2007 [ECCP07b, CCEP07]. Although serious concerns were expressed by the judges over usability,⁷ a more mature implementation (and a small security vulnerability discovered in the Prêt à Voter implementation) resulted in Punchscan being awarded the grand prize.⁸ This work formed the basis of the Master’s thesis due to Essex [Ess08]. Numerous practical limitations of Punchscan remained, however, and tackling some of these issues forms the basis of the research beginning in Part I.

Toward End-to-end Verification Without Cryptography. A number of other of non-cryptographic proposals were made in the spirit of simpler, easier-to-explain examples of the end-to-end paradigm although these schemes (like their conventional counterparts) rely on trust assumptions for correctness. Custodio introduced the floating receipt model [Cus01]. Araujo et al. later proposed variants [ACG06, AR08]. As we note later in Chapter 6, correctness is still contingent on a trusted physical chain-of-custody. Another proposal was ThreeBallot [RS07], which relies on trusted hardware components to ensure correctness. Further a number of papers pointed to vulnerabilities to passive secrecy in several scenarios [CEA07a, CKW08, HSS09] as well as coercive strategies to undermine receipt-freeness [KRMC10].

2.2.3 Concurrent Work

We now briefly review work done concurrently with this research (from the late 2000’s to present) consisting primarily of work done on in-person optical-scan schemes, with some discussion of Internet/remote voting schemes.

With regard to cryptographically verifiable optical-scan (the focus of this dissertation), the literature can be roughly separated into two categories: systems using single layer

⁷<http://www.vocomp.org>

⁸<http://www.wired.com/threatlevel/2007/07/us-team-wins-vo/>

ballot forms but reliant on trusted parties/hardware and systems with distributed trust but with multi layer ballot forms.

Single Layer Optical-scan Ballots with Trusted Components. The Prêt-à-Voter [CRS05, RS06, RBH⁺09] system and its variants [AR06, XSH08, RT09a, Rya11, CBH⁺11] also offer the voter a single-layer ballot form with randomized candidate list. Although the correctness proofs are usually described as a multi-party computation, ballot forms are generated by a trusted printer. Cast ballots are generally “encrypted” though variants exist that leave a human readable paper trail [LR08].

Benaloh [Ben08] proposes that receipts be generated and printed by a special-purpose device connected to the optical scanner. This has the distinct advantage that the ballots contain no identifying information (beyond the vote). However the issue of trusted ballot printing instead becomes a matter of trusted receipt printing.

In addition to the proposals above, several papers have found subtle potential attacks on the receipt-freeness of optical-scan based schemes under certain conditions [MN07, BMR07, CHL09, KRMC10].

Distributed Trust Optical-scan with Multi Layer Ballots. Kubiak [Kub06] and Carback et al. [CPSC07] propose *mostly* distributed modifications of the Punchscan system [PH06]. The former still relies on a trusted ballot printer; the latter distributes printing but still relies on trusted hardware to generate ballot tuples. Popoveniuc and Carback [PC10] later propose a *three*-party distributed version of Punchscan in which top-, middle-, and bottom-sheet permutations are each generated by independent printing authorities. In all cases voters must use an indirect marking procedure.

Moran and Naor [MN07] propose an improved multi layer ballot form that does not rely on indirection and with considerably stronger, provable, security properties. Voters are issued layers in separate sealed envelopes. Once inside the booth the voters are directed to remove each layer from its envelope and stack the layers in a particular order. The resultant candidate list is horizontally offset from the optical scan ovals by a randomized amount.

Lundin et al. [LTR⁺06] propose a distributed construction of the Prêt-à-Voter ballot based on a form of dealerless 2-party visual cryptography. The voter must be careful to align the VC shares in the booth in order to reconstruct the candidate list. Most recently Küsters et al. [KTV09] present a version of Prêt-à-Voter system without a trusted printer, physically implementing a re-encryption mixnet using scratch-off coatings. The voter receives a separate ballot for *each* candidate, which can be cumbersome for races involving more than a few candidates.

Other Optical-scan Schemes. Chaum proposed the first physical receipt based voting system [Cha04]. It consisted of two visual crypto layers showing the name of the voted candidate. A receipt is created by separating the layers and destroying one of them. Paul et al. [PERW03] propose visual crypto for use in voter authentication for (non-cryptographic) remote voting systems. Scratch & Vote [AR06], Scratch, Click & Vote [KZ10] and Pretty Good Democracy [RT09a] make use of scratch-off coatings to conceal encryption random factors and confirmation codes. Finally, Kelsey et al. [KRMC10] propose a voter-coercion strategy involving the use of scratch-off cards to direct voter action.

DRE-based Schemes. Although optical-scan systems remain predominant in the literature, a number of proposals have been made for fully-electronic interfaces (i.e., DRE's). Neff proposed MarkPledge [Nef04], a DRE-based protocol that was briefly spun into a business (VoteHere). Neff and Adida later proposed a number of security enhancements [AN06, AN09] as did Joaquim and Ribeiro [JR11]. Müller-Quade et al. presented a number of variants of a DRE-based scheme for which receipt-freeness is provided by a secure hardware component implementing a random number generator [BMR07, BHK⁺09, BHM⁺08].

Remote/Internet Voting Schemes. Although somewhat outside of the scope of our research, we briefly summarize advances in verifiable remote/Internet voting.

Kutyłowski and Zagorski made several proposals for code-based Internet voting [KZ07, KZ10] designed at mitigating threats to correctness and privacy in the presence of malware. Joaquim et al. also present a number of similar proposals [JR07, JRF09, JRF10]. Ryan et al. proposed an Internet voting scheme based on acknowledgment codes [RT09b, HRT10].

Adida proposed the Helios system [Adi08], an implementation of Sako and Kilian's universally verifiable scheme adapted to an Internet setting, and which does not provide receipt-freeness.⁹ Adida et al. went on to deploy a number of Helios variants in several university student elections [AMPQ09, BGP11].

Clarkson et al. [CCM08] developed an implementation of Juels et al.'s coercion-resistant Internet voting scheme. The quadratic complexity of Juels et al.'s scheme prompted a number of follow-up papers. Smith [Smi05] and Weber et al. [WAB07] presented the first proposals for a variant with linear tallying, but both were subsequently shown to not be coercion resistant [AFT07, SDW08]. A number of proposals have followed since by Araujo et al. [AFT07, AFT10, ARR⁺10], Haenni et al. [SHD10, KHF11, SHKS11, SKHS11, HS11], and others [CH11, BGR11], focusing on providing a linear tally amid various trade-offs relative to Juels et al.'s scheme. The area continues to be an active direction of research,

⁹To impress this point, Adida added a “coerce me” button to the voting interface, which emails a recipient the random factors used to encrypt a user's ballot.

although numerous and fundamental obstacles to real-world practicality remain with regard to usability, and even potentially the legality of such schemes.

2.2.4 Real-world Deployment

A number of cryptographically verifiable voting systems have been deployed previously in elections with binding results.

Remote/Internet Elections. Davenport et al. [DNW96] implemented a version of the scheme due to Fujioka et al. [FOO92] and ran an Internet-based student council election at Princeton in 1996. Herschberg [Her97] deployed a related implementation for a student council election at MIT the following year. Furukawa et al. [FMM⁺02, FMS10] deployed a scheme based on Sako and Kilian’s scheme [SK95] which has been used by NEC Corp. for running internal corporate elections since 2002. The Rijnland Internet Election System (RIES) was used in public elections in the Netherlands in 2004 and 2006 [HJP05, HJS⁺08]. Adida et al. [AMPQ09] deployed a homomorphically tallied variant of Helios [Adi08] for the elections of the Rector of Université Catholique de Louvain in 2009, and again in the Princeton undergraduate student government election. RIES and Helios are not receipt-free, and none of them are coercion-resistant.

In-person Optical-scan Elections. Bismark et al. [BHP⁺09] partially deployed Prêt à Voter in a student election at the University of Surrey in 2007, however, Surrey election officials halted the election midway through polling, reverting back to their conventional (non-cryptographic) setup. Essex et al. [ECCP07a] deployed Punchscan for a student council election at the University of Ottawa in 2007. Buckland et al. recently announced plans on behalf of the Victorian Electoral Commission (VEC) of Australia to field Prêt à Voter in future elections [BBTW12].

Part I

Interface Improvements for Voters and Election Officials

Chapter 3

Scantegrity

It needs to be a paper ballot, not a receipt that comes out of a machine—a sturdy paper ballot.

Bev Harris [Row08]

This chapter is adapted from published work co-authored with David Chaum et al. [CEC⁺08].

3.1 Introductory Remarks

Based on the feedback received during our experiences deploying with Punchscan [ECCP07a, ECCP07b], it became apparent that it would be more helpful to voters and election officials if cryptographic election verification could be offered as an *add-on* to an existing optical-scan set-up, rather than as a stand-alone system requiring new equipment, a new ballot style, and new procedures.

Our intuition was straightforward: make the ballot look as much like a traditional optical-scan ballot as possible. Doing so would potentially allow election officials to continue using their equipment, and, ideally, would allow voters to retain their mental models of how to mark and cast ballots.

To that end we present Scantegrity, an add-on to optical-scan based voting systems that combines the ideas of cryptographic E2E verifiability with a familiar paper ballot. Because of its comparability with existing optical-scan infrastructure, we believe Scantegrity can be deployed in a variety of jurisdictions at a minimal cost, while potentially dovetailing with common existing procedural requirements, such as paper audit trails and manual recounts.

For the great number of voters already familiar with optical-scan ballots, Scantegrity is the first cryptographically verifiable voting system that would not require them to re-learn how to mark and cast their ballots.

Although Scantegrity has been largely subsumed by Scantegrity II (presented in Chapter 3), we include it in this dissertation both as an introduction to the approach, as well as to underscore the difference in dispute resolution procedures (a major contribution of Scantegrity II).

Contributions. The contributions of this chapter are summarized as follows:

- Scantegrity: a cryptographically verifiable add-on system that can be interfaced with conventional optical scan voting systems allowing
 - A conventional ballot marking procedure,
 - A simple, opt-in, receipt creation procedure,
 - An unencrypted paper audit trail.
- An in-person procedure for resolving disputes over receipt transcription.

3.2 Voter Experience

The voter experience in Scantegrity is identical to that of regular optical scan systems except that the voter has the option of taking home a privacy-preserving receipt. To create such a receipt, a voter tears off a perforated corner of the ballot, called a ballot chit, which contains a serial number. In addition, as shown in Figure 3.1, the voter writes down the randomly-assigned code letter listed next to the candidate she chose. Note that in the ballot shown, the letter ‘T’ is beside the chosen candidate’s name, but on other ballots ‘T’ is associated with a candidate chosen independently (and randomly). Thus knowing that someone voted for a particular code letter does allow one to know for which candidate was voted.

After the election results have been tallied by the underlying optical scan system, election officials post a *public record* containing the Scantegrity serial numbers and chosen code letters of all the scanned ballots but not the candidate that was associated with the letter on the particular ballot. The voter can retrieve this public record, look up her serial number, and verify that the code letters she wrote down match what is in the posted record. A copy of the receipt can be made and given to third parties who can also check

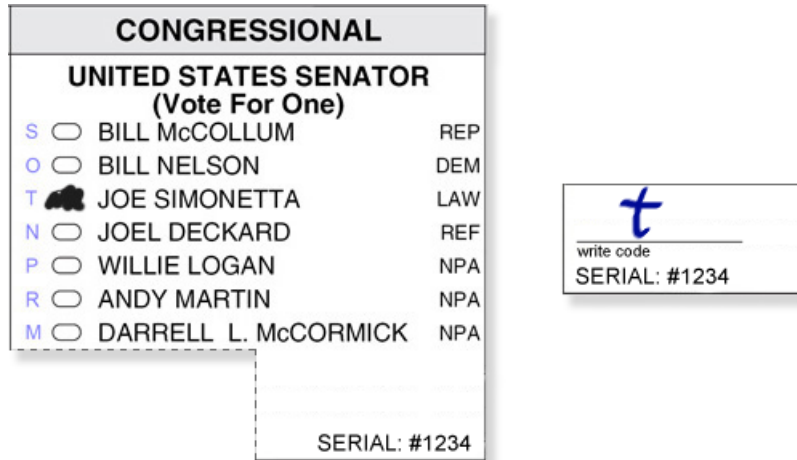


Figure 3.1: **The Scantegrity Ballot.** Scantegrity uses an optical scan ballot with randomly assigned code letters located next to each choice (left). The perforated chit in the corner contains a serial number and a space for the voter to write down a code. The chit is torn off and kept by the voter as a receipt (right). For concreteness, we use a depict a contest from the ballot of Polk County, FL in 2000.

the public record. The more receipts that are checked, the higher the chance of detecting a problem with the public record [APR07].

Correct letters indicate to the voter that the officials properly scanned and recorded her vote as she intended. However if the public record contains a letter that is different than what a voter recorded, the voter can challenge the record through the dispute resolution process (outlined in Section 3.5).

All receipts may be verified via these processes, providing proof that the votes on the ballots were recorded as they were cast. Additionally this public record is used to tabulate the results independently of the optical scan vote counting system. The next section will explain how this tabulation is done and how the result can be verified by any independent party.

3.3 The Switchboard

We have seen how the public record allows voters to use their receipt to verify that their votes were recorded-as-cast. In addition, this record can also be used by independent entities to verify the tally—that the results were counted-as-recorded. Tally verification is a challenge because the system must not directly reveal the links between code letter and candidate in order to preserve ballot secrecy.

Although Scantegrity is not the first system to provide counted-as-recorded integrity

verification, it provides, in our opinion, one of the simplest solutions to this problem. The importance of solution simplicity cannot be overemphasized in voting—it allows the widest possible audience to understand how the voting system works.

The approach of some of the E2E solutions has been to use a mix network [Cha81] to create an anonymous but verifiable link between receipt and vote. One of the properties of a mix network is that a cryptographic operation is applied at each node in the network to obscure the path of messages through it. This is especially important for identifiable (*i.e.*, unique) data such as email messages. Some E2E systems provided information on the ballot receipt (the “onion”) to allow the mix network to perform the correct cryptographic operations to count the vote correctly. Punchscan, the direct ancestor of Scantegrity, utilizes a simplified two-stage mix network with efficient cryptographic operations. It also does not require that the ballot receipt bear the onion.

We observe that election data does not necessarily need to be explicitly encrypted. Under the familiar “plurality” voting system (aka “first-past-the-post”), the voter expresses her intent by making a mark beside the candidate of her choice. A cast ballot therefore can be expressed in terms of a specific collection of marked or unmarked regions. Importantly, if treated *individually*, the states of these markable regions are not unique, and therefore need not be encrypted as they pass through an anonymizing network. Instead of using a mix network architecture, we can achieve the same anonymity properties through a simpler process—a secret permutation of the states of the markable regions (akin to shuffling a deck of cards). Thus the unique aspect of the Scantegrity component is that the permutation is used to recover the vote while hiding the link between serial number and vote.

The *switchboard* can be described as a collection of circuits established between specific markable regions on ballots in the election (*i.e.*, marked or not marked) and a particular candidate (*i.e.*, “voted for” or “not voted for”). The state of each markable region on each ballot in the election will be transmitted to votes for the corresponding candidates in the election results.

Finally, there must also be a way for the public to independently verify that marks are being transmitted through the switchboard to the correctly associated candidate without simultaneously exposing both end-points of the circuit (*i.e.*, receipt and vote).

3.3.1 Auditing the Switchboard

For voters to be confident in the switchboard’s ability to produce a correct tally, some information must be revealed for the purposes of verification. Initially when the ballots and Switchboard are created, this secret information is *committed to* using a cryptographically secure bit-commitment scheme [Ess08].

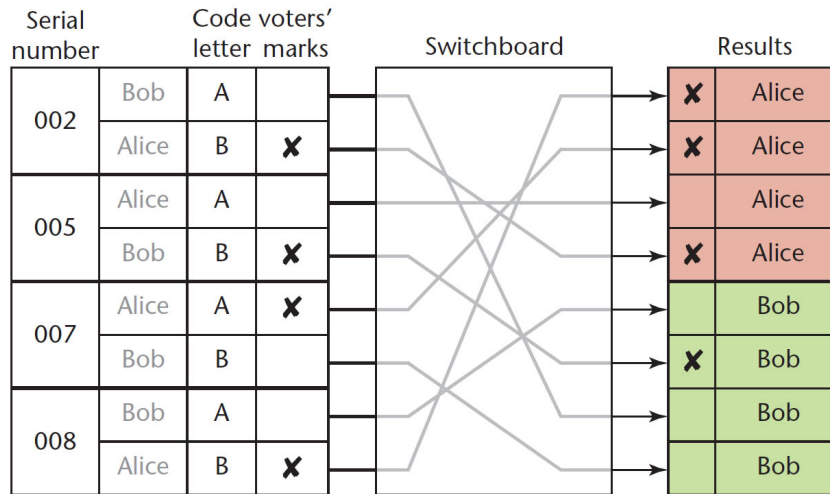


Figure 3.2: **The Switchboard.** Marks beside code letters are routed to marks beside candidates using a random and obviously generated circuit-switched network.

Before the election, these commitments are generated and published eventually allowing independent entities to verify that the secret data revealed during the audit process could not have been simply “cooked-up” on the spot. The verification requires that some secret data be publicly revealed and its correctness verified against the data committed to. We use two ways of revealing secret information:

1. Reveal the full secret and then discard it from use in the election,
2. Reveal partial information that is insufficient for revealing the secret.

We use the first technique to verify the correctness of the association between code letter and candidate in the switchboard. Before the election, half of the ballots are randomly chosen to be publicly revealed, along with their serial numbers and connections through the switchboard. Those performing this printing audit can ensure that the path through the switchboard for each candidate on each of the revealed ballots leads to a vote for the correct candidate in the results. These ballots are destroyed and will not be used in the election. If the ballots were chosen fairly and randomly, we have a high level of assurance that the remaining sealed ballots are printed and routed correctly.

After the close of polls, we use the second technique to audit the switchboard. If we segment the Switchboard into a composition of two randomly generated circuit-switched networks (*i.e.*, permutations), then revealing a link in one of the networks does not reveal the full connection. Voters’ marks travel through the first network and are recorded in an intermediary location. The marks in the intermediary position then continue through the

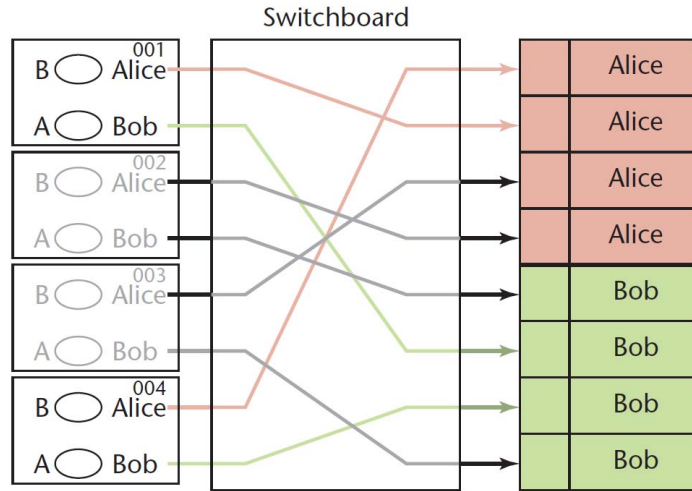


Figure 3.3: **Ballot printing audit performed before an election.** As an example we see revealed ballots #001 and #004, the association of code letters, and their connections through the switchboard. This information is made publicly available, and any independent party can plainly see a mark for Alice or Bob would have been correctly registered as a vote for Alice or Bob respectively. Once revealed, these ballots will not be used in the election.

second network to their final place in the results table. For each intermediary position, the election trustees are challenged to reveal either the link to it through the first network or the link from it through the second network but never both. Thus the connection between a recorded receipt and its position in the final results table is never revealed. For each of these links, it is publicly verified that a mark (or absence of mark) traveled through the link unchanged. In this way, observers are given a high level of certainty that the remaining secret links also routed marks correctly. To increase the statistical certainty of the audit, multiple instances of the switchboard with different random links can be used.¹

The print and mark audits outlined in this section, in conjunction with the receipt check, provide the end-to-end nature of the verification process: integrity is ensured from the printing of the ballots all the way through to the final tally.

Although these audits are conceptually simple to perform, any non-trivial-sized election would warrant the use of a software audit tool to perform these repetitive checks quickly. The software tool is intended to be open source, exceptionally easy to use, and universally available to anyone for free. Concerned parties can undertake to code their own independent version following a published specification. Ultimately, however, the software tool is only required for verifying cryptographic commitments—that the revealed audit data has not been altered. The audit itself can still be performed manually on paper ballots for

¹For each link, there is a 50 percent chance that the secret part was changed and the results successfully altered, this decreases by 1/2 for each switchboard instance.

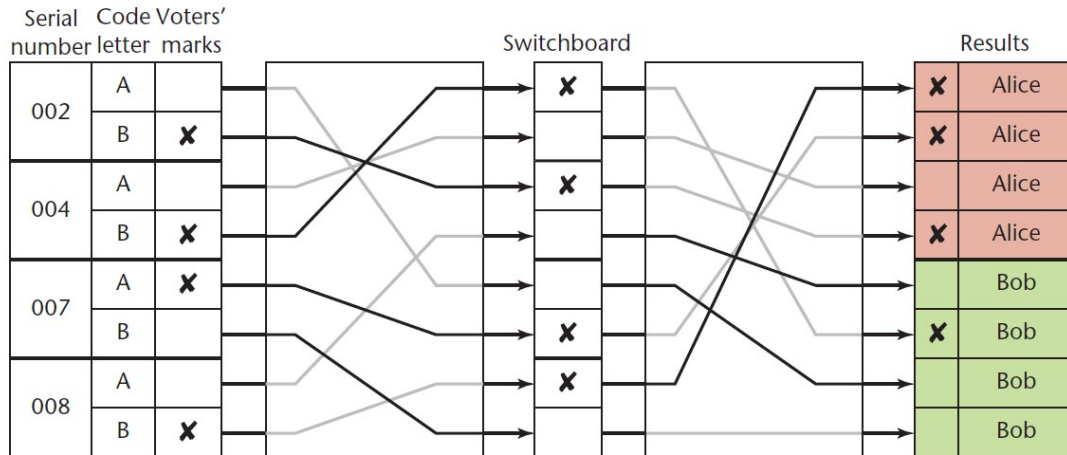


Figure 3.4: **Mark audit performed after the election.** As an example we see that ballot #002 shows a mark for the candidate with code letter ‘B’ and that this mark was correctly recorded in the intermediate position. Likewise we can see that the first vote for Alice in the results table was correctly copied from the intermediary position. Knowing only one link does not reveal the connection between code letter and candidate, preserving the privacy of the receipt.

those so inclined.

3.4 System Architecture

An illustration of how Scantegrity interfaces with the optical scan election process is shown in Figure 3.5. The election authority—a collection of election trustees—use a workstation on three separate occasions to compute all the information needed by Scantegrity. This set of “meetings” represents the three core processes of Scantegrity:

1. Before the ballots are printed, election trustees use the workstation to compute the serial number and code letters to add to the optical scan ballots as well as generate the switchboard connections. They cryptographically commit to this (secret) data and post the commitments publically,
2. After the marked ballots are scanned on election day, the *electronic ballot images (EBIs)* are given to the Scantegrity system. The code letters and corresponding voter-created marks made on each ballot are posted to the public record. Voters can check that the public record matches their receipts,
3. After the election results are tabulated and published, auditors challenge the election

trustees to open one half of the switchboard for each marking region to prove that they counted the ballots faithfully.

Using the workstation, the officials can regenerate all the data needed for each meeting, preventing the need for physically storing any sensitive election data. The workstation is secured through the removal of any external memory sources, and the open-source operating system and software is booted from a self-contained medium that can undergo attestation by anyone present both before and after its use. Fortifying the workstation is done to protect voter privacy; the integrity of the election is unconditional and thus independent of the trustworthiness of the workstation. We implemented and deployed such a workstation in a student election [ECCP07a, ECCP07b].

3.5 Resolving Disputes

Because the voter is left to record their confirmation code, it is possible that a dispute may arise between the voter and the election authority over which code appeared next to the cast candidate. Possible reasons for a dispute include voter transcription error, scanner error, or malfeasance by either party. We require a procedure, therefore, to convince both parties, as well as the broader public, as to which party is at fault. This procedure, furthermore, should not reveal the voter's preference. We achieve this in a partial sense: while the public does not learn the voter's preference during the execution of the dispute resolution procedure, the election authority does.²

To protect ballot secrecy, the dispute resolution procedure proceeds in two phases. First the election authority proves that it is in physical possession of the voter's ballot. Second, the election authority proves which code letter appears beside the voter's preference. The procedure requires the voter to have retained their receipt chit.

In the first phase, the election authority retrieves the original ballot and places it in a privacy sleeve that only shows the ballot serial number and chit perforation (see Figure 3.6(a)). The voter can then compare the perforation pattern of their chit against the ballot. If it is necessary, forensic analysis can be performed to match the fibers of the chit to the ballot.

In the second phase, the election authority moves the ballot to a second privacy sleeve that only shows the optical-scan ovals and confirmation codes. The official notes the position of the marked letter and drops the sleeve into an empty lottery-style hopper. The election authority then collects a set of dummy ballots with the same code letter marked

²In Chapter 10 we describe how the dispute resolution procedure can be extended to hide the voter preferences from the election authority.

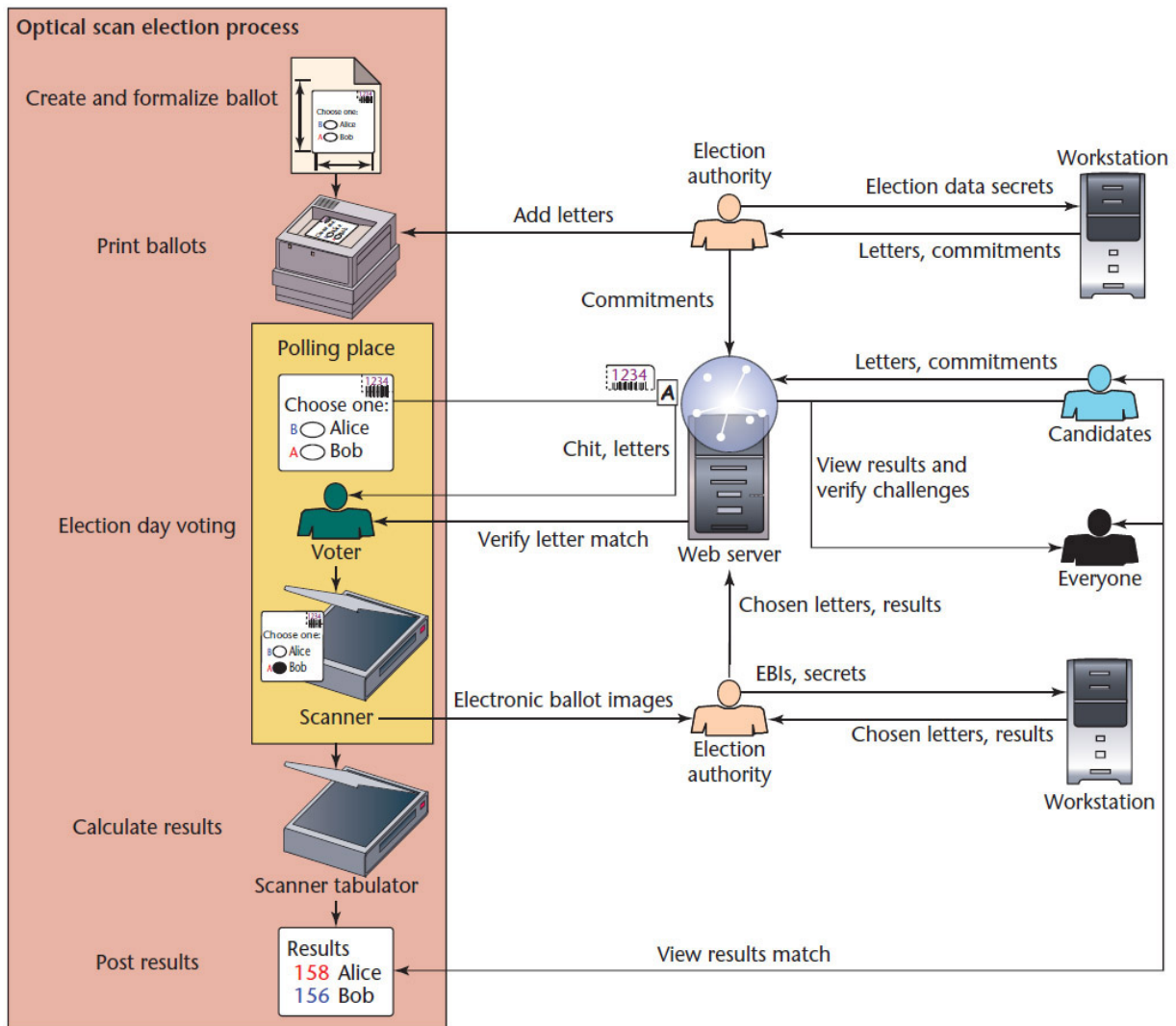


Figure 3.5: **Overall Election Process with Scantegrity.** Scantegrity inserts itself into the traditional optical scan election process, creating a separate mechanism for independent universal verification of election results. Election trustees use a workstation to create letters and add them to the ballots. After voting, they use the workstation again, reading the electronic ballot images (EBIs) to interpret marks on each ballot and post the chosen letters. They use the workstation a third time to respond to audit challenges, and everyone can check the responses to be sure the results decrypted properly.



(a) Proving Possession of Ballot

(b) Proving Code Letter Marked

Figure 3.6: **Scantegrity Dispute Resolution Procedure.** Left: the receipt chit is physically matched to the originating ballot (sheathed in a special privacy envelope/sleeve). Right: the original ballot (and several dummy ballots obscuring candidate preference) prove which code was marked.

for each of the other candidates, puts them in similar privacy sleeves, and drops them into the hopper. After tumbling the hopper, the election official retrieves each privacy sleeve envelope and places it in plain view (see Figure 3.6(b)). Since the ballot was already matched to the chit, it is guaranteed to be in this collection. Thus the election officials have successfully demonstrated what code letter was voted for without revealing the candidate that was voted for. The single code letter (marked on all the shown ballots) can then be compared to the public record. After all disputes are settled, everyone can assume that the public record of chosen letters is correct, and that no ballots were lost. If necessary, officials re-compute the results from the corrected public record.

A physical in-person dispute resolution procedure is obviously inefficient and does not scale well. The major contribution of the next chapter will be to develop an informational dispute resolution procedure for Scantegrity.

3.6 Implementation

We have created a Java-based software implementation and merged it with the Punchscan codebase. Our software is general enough to author ballots of both styles, even allowing an election to mix Punchscan ballots with Scantegrity ballots. The software takes a ballot layout in PDF format as input and produces a multiple-page PDF document of the ballot collection with letters and serial numbers inserted.

We tested our implementation with both moderate and large sized elections. On a 1.73 GHz laptop, we were able to tabulate 1 million ballots in under 10 minutes. Using actual statistics from Florida's 2000 Polk County election,³ where there were 32 contests with an average of 3.2 candidates per contest, we tabulated 200,000 ballots in under 4 minutes, slightly more than the number of ballots that were cast in that election. Auditing the results took less than 2 minutes.

3.7 Concluding Remarks

Scantegrity is a potentially attractive solution to the problem of real-world trustworthy voting. In addition to a universal cryptographic end-to-end verifiable result, it offers an opt-in procedure for receipt creation, which, as a crucial benefit, allows voters to mark their preferences vote using a familiar paper ballot. Furthermore, the add-on nature of Scantegrity provides an opportunity for election officials to largely retain their existing optical-scan equipment and polling place procedures. It also leaves officials with a conventional (i.e., human-readable/unencrypted) paper record.

As such, Scantegrity comes much closer to meeting existing laws and requirements of many jurisdictions than its predecessors. The in-person dispute resolution procedure, however, remains a major limitation of this approach. In the next chapter, we will tackle the problem of automating the dispute resolution process.

³Election website for Polk County, Florida. Retrieved in 2007. <http://www.polkelections.com/>

Chapter 4

Scantegrity II

The purpose of any election system is to provide sufficient evidence to convince the loser of an election that he or she has genuinely lost.

ACCURATE research
proposal [RWB⁺05]

This chapter is adapted from published work co-authored with David Chaum et al. [CCC⁺08, CCC⁺09]

4.1 Introductory Remarks

The voter-created receipts introduced by Scantegrity are advantageous from a usability perspective owing to their opt-in nature: voters can complete their primary task (*i.e.*, vote) without learning new procedures. As a major disadvantage of this approach, however, and in contrast to Punchscan and Prêt à Voter, receipts are not automatically created by the act of marking a ballot, requiring instead a conscious action on behalf of the voter (*i.e.*, recording a confirmation code). Furthermore, because receipt creation is unsupervised, it is possible for disputes to occasionally arise between voters and election officials over which confirmation code is correct. Scantegrity II addresses this latter point in a more sophisticated manner than its predecessor.

A cumbersome in-person dispute resolution protocol was proposed in the previous chapter (see Section 3.5). In this chapter we present Scantegrity II, a variant of Scantegrity

that allows disputes to be handled through an online (as opposed to in-person) protocol. The approach to an online dispute resolution procedure is twofold: one is to increase the code space so as to make guessing valid codes statistically unlikely. The other is to only reveal to the voter the codes corresponding to their selections.

In Scantegrity II, voters mark ballots using a special ballot-marking pen, which makes legible pre-printed confirmation codes corresponding to voter selections. The link between confirmation codes and voter selections is cryptographically protected, with the key(s) being shared by election officials. Voters may note down their confirmation codes onto a chit that is detachable from the ballot. After the election, all voted confirmation codes are posted online, where voters may check them. The final tally is computed in a verifiable manner from the posted confirmation codes.

The functionality of Scantegrity II is enabled by the use of invisible ink, in the following ways:

- Confirmation codes and ballot ovals are printed with invisible ink, which darkens when it reacts with the ink in the ballot-marking pen; the confirmation code ink reacts more slowly than the ballot oval ink, and hence darkens several minutes after the oval does. Thus, the code is visible for several minutes after being marked, during which the voter may note it on the chit. On the other hand, the confirmation code may be assumed to be indistinguishable from its background in an unmarked oval. This allows the Scantegrity II system to provide a confirmation code to the voter only after the voter has made the corresponding ballot selection.
- The Scantegrity II chit bears two serial numbers that are required of the voter in order to check the confirmation codes online. These serial numbers are also indistinguishable from the background until made legible through the use of a decoding pen. The ink in the decoding pens is different from the ink in the ballot-marking pens. Poll workers reveal the serial numbers using a decoding pen after the ballot is cast. This prevents voters from falsely claiming that a valid confirmation code, obtained from an uncast ballot, came from a cast ballot. When it is not possible to use the different inks required for chit serial numbers and decoder pens, it is possible to achieve a similar end, though with weaker integrity guarantees, by requiring that a record be kept, by polling officials and observers, of serial numbers of spoiled ballots.

Contributions. The contributions of this chapter are summarized as follows:

- An improved, informational, dispute resolution procedure based on voter knowledge of a confirmation code,

- An invisible ink printing process for physically disseminating confirmation codes to voters,
- A ballot with closely related marking and scanning procedures to that of Scantegrity (and conventional optical-scan) ballots.

4.2 Scantegrity II Procedures

Scantegrity II provides integrity guarantees through the use of a confirmation code provided to each voter for each ballot selection. All confirmation codes are posted on a website after the election, and all results are obtained through the processing of these codes. The Scantegrity II protocol defines the manner in which participants in the election—voters, election administrators, and observers—interact with the voting system in order to ensure that (i) confirmation codes are correctly present on the ballots, (ii) marked confirmation codes are correctly present on the website, and (iii) confirmation codes are correctly processed to obtain the final tally. The protocol is designed to enable the detection of election fraud if it has occurred, as well as to prevent false charges of election fraud. This section provides an (intentionally) informal description of the protocol; its purpose is to provide a description that is somewhat accessible to voters, poll workers, and election administrators, and to prepare the reader for the more formal description in the next section.

4.2.1 The Vote Casting Procedure

This section describes the vote casting procedure, which is very similar to that of a regular optical scan ballot. The slight differences between the two are as follows: first, the unmarked ballot itself looks slightly different: it bears a detachable chit that can be used to note confirmation codes. Second, while marking the ballot, voters will notice the appearance of confirmation codes, which will also disappear after a few minutes. Third, voters or observers may *audit* ballots to determine whether printed confirmation codes correctly reflect voter selections; such ballots may not then be cast. While we have simplified the ballot audit procedure considerably, it does not have a corresponding equivalent in the regular optical scan protocol, and might appear complicated to voters and officials. Similarly, spoiled ballots are discarded using a procedure that is more complex than that used for optical scan. Fourth, voters interact with a polling official after the vote is successfully cast, in order to expose serial numbers on the receipt chit.

The Scantegrity II Ballot. The Scantegrity II ballot consists of two parts: the *main body* and the *chit*; see Figure 4.1. Similar to an optical scan ballot, the main body of

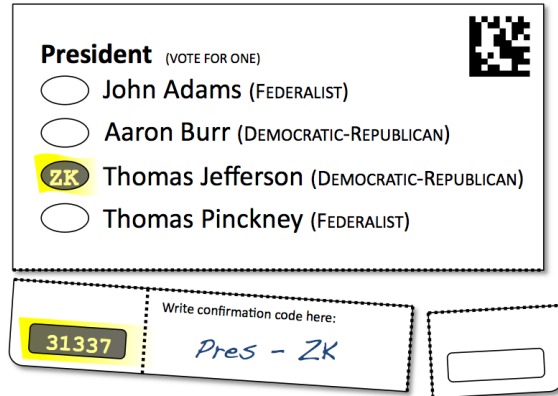


Figure 4.1: A Scantegrity II ballot showing the main body (top) with one marked position and machine-readable serial number; left chit (bottom left) with a developed chit serial number and confirmation code written in; and right chit (bottom right) with an undeveloped chit serial number. This figure is meant to demonstrate the parts of the ballot and does not represent the actual final state of the portions after voting.

a Scantegrity II ballot contains, for each contest, a list of valid selections printed in a canonical order pre-determined by polling place procedures (*e.g.*, alphabetical, rotated across precincts, *etc.*). Next to each possible selection is a markable region, oval in shape.

Differing from an optical scan ballot, the background of each oval is printed with a reacting ink. The confirmation code corresponding to the selection for the particular ballot is printed inside the oval. The ink used to print the confirmation code is similar to that used for the oval background, but is slow-reacting. Both inks look the same when printed on the ballot paper; they may be assumed to be indistinguishable to the human eye before the oval is marked with the ballot-marking pen (see Section 4.5 for details on the validity of this assumption). Further, we assume that voters will not be able to take expensive spectral analysis equipment into the polling booth; such equipment might aid in the ability to distinguish between background and confirmation number. Thus, we assume that, before marking, the oval has a single color, and confirmation codes are indistinguishable from the background of the oval; that is, confirmation codes are invisible. Additionally, a Scantegrity II ballot contains a ballot serial number that is machine-readable but not easily read or memorized by a human (*e.g.*, a two-dimensional barcode).

The chit is attached to the bottom of the ballot via a perforation, such that it can be easily detached. It has two halves, left and right; the halves can be detached from each other using a pair of scissors. On each half is a chit serial number: the left chit serial number and the right chit serial number. These chit serial numbers are distinct from each other and from the ballot serial number; we describe later how they are used to ensure that voters cannot make false claims regarding confirmation codes on uncast ballots. Both

the left and right chit serial numbers are printed in invisible ink such that they are neither human nor machine readable before being decoded using a special decoder pen. Both the left and right chit serial numbers are assumed to appear after they are marked with the decoder pen.

Ballot Marking. Upon arrival, a voter is authorized to cast a ballot, and is handed the next one in the pile; it is enclosed in a privacy sleeve. At this time, she may choose to audit a ballot, which she may choose from the existing ballot pile. For details on the ballot audit procedure, see Section 4.2.2.

In order to vote for a particular selection, the voter fills in the corresponding oval using a ballot-marking pen. In accordance with the invisible ink printed on the ballot, the background of the oval will immediately turn dark, leaving a confirmation code visible in the foreground. The relative darkness of any marked ovals to unmarked ones will allow an optical scanner employing dark mark logic to register the oval as marked. The foreground of the oval will be human-readable and a voter interested in verifying that her vote is in the virtual collection of votes to be tallied may record the code on the chit portion of the ballot. Uninterested voters may disregard the codes.

The link between a confirmation code and the corresponding selection on a particular ballot is protected cryptographically. We omit the details underlying the generation and protection of the data until the next section. At this stage, however, we do note the following: the disclosure of a confirmation code does not reveal the selection, if the cryptographic techniques used are assumed secure, and election officials are assumed not to collude to determine the selection.

Although not apparent to the voter, the confirmation code is printed in a slow-reacting invisible ink that will also turn dark, but only after the passage of several minutes (*e.g.*, five to seven minutes). At this time, the oval will be completely dark and the code will no longer be visible, leaving no human-readable unique information on the ballot.

As an option, the two-dimensional bar-coded serial number could also have slow reacting ink in its background such that if a voter marked it, it would turn solid black.

Section 4.5 describes how a masking ink and appropriate printing techniques may be used to reduce the ability to distinguish between the inks, even with the use of microscopes and spectral equipment. Indeed, it may be assumed that the slow and fast-reacting invisible inks are, for all practical purposes, indistinguishable (i) before exposure and (ii) T seconds after both have been exposed, where T is the response time of the slow-reacting ink. After a period long enough to include reaction times, a filled-in Scantegrity II ballot provides, for all practical purposes, an amount of information that is similar to that on an optical scan ballot, and can be used in a manual recount with a level of privacy very similar to that of optical scan.

Spoiling the Ballot. If the voter makes an error in marking a ballot or wishes to register a protest vote through spoiling the ballot, it is returned to the poll worker. Without seeing the contents of the ballot, the poll worker removes the ballot from the privacy sleeve and detaches the right side of the chit from the ballot. The main body and left chit are shredded in view of the voter. The right chit is retained by the poll worker and used to verify that the number of ballots issued is identical to the sum of the number of ballots tallied, print-audited, and spoiled. The number of spoiled ballots allowed per voter is typically limited by pre-determined polling place procedures.

Casting the Ballot. When the voter has satisfactorily marked a ballot, it is returned to the poll worker. As previously, the poll worker detaches the chit from the ballot. Further, with the choices on the ballot still concealed, the poll worker places the main body of the ballot into the scanner, which records the ballot serial number and the marked choices. In the preferred version of the protocol, voters are not allowed to cast undervoted or overvoted ballots. If a voter does not wish to vote for a particular candidate, she must make a selection of “none of the above”. If the scanner detects an undervote or overvote, the voter is returned her ballot, and will spoil it and re-enter the issuance procedure. Note that, in the US, the requirement that a voter be notified of undervotes or overvotes is not uncommon; in fact the Help America Vote Act requires that voters be notified of overvotes if electronic equipment is used. However, requiring that undervoted or overvoted ballots not be cast is considerably stricter, and decreases the usability of the voting system. The alternative version of the protocol does not ban undervotes or overvotes in cast ballots. However, in this version, a secure chain of custody is required to ensure that unvoted races were not changed to voted ones, nor voted races overvoted.

In order for the scanner to read the serial number, it must be encoded in a two-dimensional barcode as the scanner can only recognize marked or unmarked regions.

After a successful scan, the two serial numbers on the chit are developed by the poll worker. The voter may leave with the chit. It is expected that public interest groups will make available the possibility of creating a copy of chits to alleviate the need for concerned but time-constrained voters to personally participate in auditing the election.

Casting without Automation. For polling places without adequate voting technology or in the event of a power failure, Scantegrity II may still proceed with the voter being issued the chit in the same manner. The main body of the ballot will, instead of being scanned, be placed into a sealed ballot box that has been certified as being empty prior to sealing. If scanning technology is unavailable at the polling place, the ballots may be transported to a central scanning location.

Accounting for Ballots. At the end of the day, poll workers and official observers make a note of the numbers of spoiled, voted and audited ballots, and ensure that their sum is equal to the number of used ballots. These numbers are made publicly available; this prevents ballot stuffing. Further, they note down the exposed chit serial numbers of voted, spoiled and audited ballots, so these cannot be changed after the election.

4.2.2 Election Audit Procedures

A voter may participate in auditing the election in several ways. In addition to checking the confirmation numbers on her ballot, she may audit a printed ballot, and check the processing of confirmation codes. Election observers may also participate in the latter processing check.

Auditing a Printed Ballot. Voters wishing to audit a printed ballot may choose one from the ballot pile; we refer to the process of auditing the ballot as the *print audit*. They will each be issued a ballot main body and the left or right half of the chit, with the serial number activated using the decoder pen; which half is chosen may be determined by a flipped coin. The other half of the chit is removed and retained by the pollworker in a clear box on the poll worker table. At her leisure, the voter fully marks the ballot to reveal all the confirmation codes, which she may check using the procedure in the following section.

Checking Confirmation Numbers. At a pre-arranged time after the polls close, voters who recorded the confirmation codes associated with the candidates they voted for, or those who wish to check the confirmation codes on a print-audited ballot, may visit a website where they will be prompted for the serial number on the chit. In the case of voted ballots, the voter will have two serial numbers—left and right; either is suitable to identify the ballot uniquely. Upon entering a serial number, the website will report the confirmation codes in the positions it believes were marked for voted ballots, but will not report the candidates associated with these codes. For this reason, providing a copy of the confirmation codes in no way undermines the secrecy of the ballot. Voters are encouraged to share their confirmation codes, share photographs of their chits, or post screen-captures of the results. In the case of an audited ballot, entering the serial number will similarly report the confirmation codes that should appear on the ballot and, only in this case, also reveal the candidates associated with each code.

All confirmation codes and their associated candidates are committed to prior to the election to ensure the values or associations cannot be changed. Thus, the audited ballots provide probabilistic evidence that the confirmation codes were correctly printed on the

ballots. The correct and full inclusion of confirmation codes from a voted ballot provides probabilistic evidence that the votes were properly scanned and not maliciously altered. Full details are provided in Section 4.3, and the strength of this evidence is quantified in Section 4.4.

Checking the Processing of Confirmation Numbers. Due to the commitments to confirmation codes and candidates before the beginning of the election, it is known that candidates are mapped to confirmation codes and that this mapping cannot be changed. Further, through the print audits, voters are assured that this mapping has been faithfully transposed to the printed ballots they marked. By checking the inclusion of their confirmation codes, they are further assured that the marks they made for candidates have been faithfully transposed to confirmation codes consistent with those on the ballot. The final step is to check that the confirmation codes are properly mapped back to the correct candidates.

The protocol for achieving this check will be based on an open specification. Voters may either obtain software from a software provider they trust, or write their own software, to check the processing of the confirmation numbers. All required information for writing the software (such as the format of the data and what the data are) is provided by Scantegrity II to all interested parties. Those administering the election are encouraged to appoint an independent auditor to perform this check so as to provide at least one audit of the tally computation from confirmation codes. The details of this check are also provided in Section 4.3.

4.2.3 Dispute Resolution Process

If any voters discover incorrect confirmation codes or ballots that are incorrectly designated as voted, print-audited, or spoiled, they may file disputes. In the case of a confirmation code being incorrect, they may provide the confirmation code they believe should be on the ballot. A voter's knowledge of a valid confirmation code on the ballot that is not present on the website, suggests an error or malfeasance; the validity of the code can be established since the codes are committed to, and the likelihood of guessing a correct code can be made low through the use of longer codes (exact quantification to follow in Section 4.4). If a voted ballot is incorrectly designated, the voter can provide both chit serial numbers to prove that it was voted. Similarly, if a print-audited ballot is incorrectly designated, the voter or independent auditor can provide all the confirmation codes on the ballot to prove that it was print-audited. In the case when the voter knows all confirmation codes in an overvoted ballot, this ballot's designation cannot be changed to print-audited as the voter knows both serial numbers. In order to ensure that unvoted races are not voted, and

that properly voted ballots are not changed to overvoted ones, a restriction of not allowing undervotes or overvotes on cast ballots is required.

4.3 Cryptographic Proof of Tally

The following describes the method used for proving the correctness of an election outcome while simultaneously maintaining voter anonymity. It is based on the protocol described in Chapter 3, adapted to the enhanced polling procedures described in Section 4.2.

4.3.1 Ballot Definition

For simplicity we consider a notation based on a single contest ballot. Generalization to ballots containing multiple races, as well as elections containing multiple ballot styles, should be viewed as multiple independent executions of the single contest case described herein. Let $L = (s_0, \dots, s_{n-1})$ define a list of n ballot selections (e.g., candidates, choices, etc).

4.3.2 Roles

We consider three categories of entities participating in the election with the acknowledgement that the entities are role-based and thus an individual might possibly assume any or all roles.

Voters: Voters are those with the authority to cast a ballot in the election. We assume that voter authentication (external to this discussion) is undertaken prior to ballot issuing and that only authenticated voters are issued ballots. In this section we will refer to a particular voter as V .

Election Trustees: Let T be the set of t election trustees, $T_1, \dots, T_t \in T$. The trustees engage in the cryptographic protocol to setup and generate the correctness proofs of the election. T would generally consist of public officials and, optionally, candidate representatives. The protocol is intended to proceed when a minimum number of trustees are present—not requiring the presence of all so as to mitigate the disruption caused by any individual trustee’s absence at various stages of the protocol.

Verifier: The set of verifiers A consists of all agents verifying the correctness proofs herein. The intention is that the tally-correctness be “universally verifiable” as defined in [SK95]—meaning that any voter, citizen or observer can participate either directly, or through delegation, in the verification of the tally if they so choose.

Other Entities: Poll workers are responsible for administering the voting process, instructing and assisting voters, as well as enforcing the registration, ballot issuing, marking and casting procedures outlined in the previous section.

Finally, we require the existence of a public bulletin board \mathcal{BB} that implements an append-only public record. In practice it might be implemented as a mirrored public website.

4.3.3 Functions

In this section, we outline the main functions used in the protocol. For a positive integer len , we use $[len]$ to denote the set of integers $[0, 1 \dots len - 1]$.

The functions consist of:

1. A parameter initialization function that, given a security parameter, provides an election-specific nonce and minimum key lengths.
2. A trustee threshold-key generation function that produces individual trustee keys for trustees and a master key that can be reconstructed from a minimum threshold τ number of trustee keys. This function takes as input the election-specific nonce, the value of τ , and input bit strings from the trustees, the entropy of which provides the entropy of the keys generated.
3. A master-key reconstruction function that, given a set of τ or more trustee keys, reconstructs the master key.
4. A subkey generation function that is a cryptographic one-way function, accepts a master key and an identifier, and outputs another key.
5. A keyed permutation function that, given a key and the value len , generates a pseudo-random permutation of integers in the range $[len]$.
6. A cryptographic commitment function that is computationally hiding and computationally binding.
7. A ballot generation function that, given the candidate list, the confirmation code alphabet and length, the election master key, and the number of ballots required, generates the master list of ballots.

Details of each of these functions follow.

Parameter Initialization: $P \leftarrow \text{Parameters}(1^p)$ accepts a security parameter p and outputs a set of functional parameters P including a unique election-specific nonce λ

selected in accordance with a public convention (not considered here), and a specification of cryptographic algorithms used to realize certain cryptographic one-way and trapdoor functions, as well as specifying their enforced minimum key lengths. For brevity, we will omit continual reference to P by assuming all following functions accept it as input.

Trustee Threshold-Key Generation: $(k_1, \dots, k_t, K) \leftarrow \text{TrusteeKeys}(\omega_1, \dots, \omega_t, \tau, \lambda)$ accepts an arbitrary-length random bit string, denoted $\omega_i \in \{0, 1\}^*$, from each trustee T_i , as well as a threshold $1 \leq \tau \leq t$, specifying the number of trustees needed to reconstruct a unique election master key K . It outputs a distinct key for each of the trustees, k_1, \dots, k_t , as well as a master key K . We do not consider the policy guidelines for selecting trustees or τ in this section. K is such that, if at least one ω_i is uniformly distributed across all possibilities, K will be as well. K is also dependent on the election nonce λ (so if the same value of ω_i were supplied in a different election, K would be different). K is only used as private input to other functions. Each output key k_i is transmitted over an authenticated and physically untappable channel to the corresponding trustee T_i .

Election Master Key Reconstruction: $\emptyset/K \leftarrow \text{ElectionKey}(\{j_1, \dots, j_n\})$ accepts as input a set $\{j_1, \dots, j_n\}$ of keys and outputs the unique election master key K if and only if $|\{j_1, \dots, j_n\} \cap \{k_1, \dots, k_t\}| \geq \tau$. Otherwise it returns a symbol (denoted by \emptyset) indicating the function failed to reconstruct the key.

The assumption for the two preceding algorithms is briefly stated: given any unbounded adversary \mathcal{A} , the advantage of \mathcal{A} (over a random guess) in guessing K , given any set containing fewer than τ keys from k_1, \dots, k_t , is exactly zero. One suitable construction is due to Pedersen [Ped91], and has been suggested for use in voting by Benaloh [Ben06]. A suitable notion of an untappable channel is the one due to Sako and Kilian [SK95].

Sub-key Generation: $\kappa_{ID} \leftarrow \text{SubKey}(K, ID)$ is a cryptographic one-way function that accepts a master key K and identifier ID and outputs another key κ_{ID} , where ID defines what key is to be generated.

Keyed Permutation: $\pi \leftarrow \text{Perm}(\kappa, len)$ accepts a key κ and list length len , and outputs $\pi : [len] \rightarrow [len]$ where π is a permutation selected pseudo-randomly from the set of possible permutations of len elements Π_{len} . The function π depends on κ . We use the notation $X'(i) \leftarrow X(\pi(i))$ to denote the element-wise shuffle of a len -element set X for $0 \leq i < len$. Finally, we define a special-case null index, denoted \emptyset , in which $\pi(\emptyset) = \emptyset$ for all $\pi \in \Pi_{len}$.

Cryptographic Commitment: We consider a cryptographic commitment protocol as including the following pair of functions: $\bar{m} \leftarrow \text{Commit}(\kappa, m)$ accepts a key κ and an arbitrary length message m to obtain a commitment \bar{m} . $0/1 \leftarrow \text{Decommit}(\kappa', m', x)$

accepts a commitment x , key κ' , and message m' , and outputs 1 if $\text{Commit}(\kappa', m') = x$. Otherwise it outputs 0.

The cryptographic assumptions for these algorithms are briefly stated: given any probabilistic polynomial time-bounded \mathcal{A} producing messages m and m' , and keys κ and κ' , the probability that $m \neq m'$ and $\text{Commit}(\kappa', m') = \text{Commit}(\kappa, m)$ is a negligible quantity in the security parameter p . That is, \mathcal{A} cannot find two distinct messages that produce the same commitment. This is an informal definition of the computationally binding property of a commitment. Additionally, given any probabilistic polynomial time-bounded \mathcal{A} ,

$$|Pr[\mathcal{A}(\text{Commit}(\kappa, m)) = 1] - Pr[\mathcal{A}(\text{Commit}(\kappa, m')) = 1]|$$

is a negligible quantity in the security parameter p . That is, \mathcal{A} cannot distinguish between a commitment to m and one to m' , if the commitments use the same key. This is an informal definition of the computationally hiding property of commitment functions.

Generate Ballots: $\mathbf{P} \leftarrow \text{GenerateBallots}(L, \Sigma, \ell, K, b)$ accepts ballot selection/candidate list L of size n , confirmation-code alphabet Σ (typically the set of alphanumeric characters), confirmation-code length ℓ , election master key K , and the overall number of ballots to be generated b . \mathbf{P} contains three lists. The first is a list of b ballots, sorted by serial number, each with n selections, each selection associated with a confirmation code in Σ^ℓ . In addition to this list, \mathbf{P} also bears space for the voters' choices after ballots are filled, and a third list which bears the corresponding candidates.

We deviate slightly from the notation introduced in [CCC⁺08]. Let \mathbf{P} denote the canonical “master” list associating codes, candidates, and voter-made marks, which we define as the triple of $(b \cdot n)$ -element lists $\mathbf{P} = \{\mathbf{Q}, \mathbf{R}, \mathbf{S}\}$. For all $0 \leq j < bn$,

1. \mathbf{Q} is a list of serial numbers and confirmation codes, including serial numbers (α, β, γ) for each ballot, and confirmation codes q for each selection in a ballot. Let $\mathbf{Q}(j) = \{\alpha_j, \beta_j, \gamma_j, q_j\}$,
2. \mathbf{R} will eventually represent the list of scanned voter-made marks $r \in \{0, 1\}$ indicating the absence or presence of a mark (*i.e.*, vote) made for an associated selection. Let $\mathbf{R}(j) = r_j$, and let all r_j be initialized to 0,
3. \mathbf{S} is a list consisting of b repetitions of selection/candidate list $L = (s_0, \dots, s_{n-1})$. Let $\mathbf{S}(j) = s_{(j \bmod n)}$.¹

For notational convenience throughout the rest of this chapter we will use the index g to refer to a given ballot B_g , and its associated voter-receipt V_g where $g = \alpha_j = \lfloor j/n \rfloor$. For any $j \neq j'$ let $\alpha_j = \alpha_{j'}$, $\beta_j = \beta_{j'}$, $\gamma_j = \gamma_{j'}$ if $\lfloor j/n \rfloor = \lfloor j'/n \rfloor$.

¹Note that this construction differs slightly from that of Chapter 3.

Serial numbers β, γ shall be selected independently (without replacement) by a secure pseudorandom number generator seeded by the election master key K . These numbers shall be selected from range defined by p , such that correctly guessing an unknown β or γ would occur with a small (but not cryptographically negligible) probability.

Finally, confirmation codes q will be independently selected by a pseudorandom generator such that confirmation codes are not repeated across a given ballot B_g , namely $q_j \neq q_{j'}$ if $\lfloor j/n \rfloor = \lfloor j'/n \rfloor$, for distinct j, j' .

See Figure 4.2 for an example of a list of four ballots when there are two candidates on the ballot, and confirmation codes consist of three alphanumeric symbols.

j	α	β	γ	q
0	0000	7973	4630	7LH
1	0000	7973	4630	WT9
2	0001	2567	1490	J3K
3	0001	2567	1490	TC3
4	0002	4900	7891	9JH
5	0002	4900	7891	J3K
6	0003	1631	5275	KWK
7	0003	1631	5275	H7T

(a) Table **Q**

j	r
0	0
1	1
2	0
3	1
4	1
5	0
6	0
7	1

(b) Table **R**

j	s
0	Alice
1	Bob
2	Alice
3	Bob
4	Alice
5	Bob
6	Alice
7	Bob

(c) Table **S**

Figure 4.2: An example of the (private) Scantegrity II master list **P** associating codes, candidates, and voter-made marks. Tables **Q**, **R**, and **S** when there are two candidates, $s_0 = \text{Alice}$ and $s_1 = \text{Bob}$. For example, a vote for **Alice** on Ballot 0000 would reveal the confirmation code 7LH, however one for **Bob** would reveal WT9. Note that, for purposes of illustration, we show one way in which the **R** table may be populated based on votes cast during the election. The function `GenerateBallots()` however, initializes all these values to zero. In this example, the votes cast on Ballots 0000, 0001, 0002 and 0003 were for **Bob**, **Bob**, **Alice** and **Bob** respectively, and would reveal confirmation codes WT9, TC3, 9JH and H7T respectively. The publicly published versions of tables **Q** and **S** will contain commitments to the information shown above, this detail is provided in Step 6c of the setup phase in section 4.3.5. There is no information in Table **R** before votes are cast, and there is no information made public about this table before the election.

4.3.4 Trusted Computation Platform

The protocol assumes the existence of a hardware device, referred to as the *trusted computation platform*, which the trustees use to evaluate the various functions described above. This device relies on the following assumptions related to the preservation of ballot secrecy:

- **Private and authenticated input:** the ability to receive input from authenticated trustees via a physically untappable channel,

- **Private evaluation:** the ability to evaluate a function such that the intermediate values cannot be recovered by passive or active attack of the hardware or software components,
- **Correctness:** the ability to attest that the functions being evaluated are equivalent to available and predefined source code.

Note that the correctness assumption enables the trustees to be certain that the required computations are being computed correctly, and hence increases the reliability of the computation from the perspective of the honest trustee. It does not affect the ability of the voter or the auditor to detect a cheating trustee.

With the failure of any of these trust assumptions, it may become possible for a malicious subset of trustees to recover information related to the association between voting intent and ballot serial number. For example, this can be accomplished by observing a sufficient number of trustee keys, observing intermediate state, or altering the functions to overtly or covertly leak this information.

None of these assumptions, including the correctness assumption, dictates the soundness of the tally. In the event that any or all of these assumptions are subverted (or any cryptographic assumption is found not to hold), the correctness of the final tally can still be ascertained through the independent verification mechanism described in this section.

4.3.5 Protocol

Setup Phase. The trustees in set T generate their threshold trustee keys and initialize the bulletin board \mathcal{BB} using Candidate list L , security parameter p , number of ballots to be generated b , valid trustee threshold list τ , code alphabet Σ , code length l , and a heuristic security parameter I where $\{L, p, b, \Sigma, l, \tau, I\}$ is issued to T by an external entity not considered herein. The voting system commits to several proof instances, with the audit checking for consistency between them. I is the number of proof instances constructed by the system.

Let the notation $X_i(j) = x_{i,j}$ denote the j -th element in the i -th instance of a shuffled list X_i . Additionally let the notation X'_i , X''_i and X'''_i denote list X shuffled by the composition of permutations $(\pi_{(i,1)})$, $(\pi_{(i,2)} \circ \pi_{(i,1)})$ and $(\pi_{(i,3)} \circ \pi_{(i,2)} \circ \pi_{(i,1)})$, respectively.

Using a trusted computing platform, the trustees perform the following computations:

1. Initialize security parameters: $P \leftarrow \text{Parameters}(1^p)$,
2. Initialize bulletin board: Post $\{P, L, p, b, \Sigma, l, \tau, I\}$, and the specification of all functions to \mathcal{BB} ,

3. Generate trustee keys: Each trustee T_i contributes entropy ω_i and is issued corresponding trustee key k_i via an untappable channel with the trusted computing platform $(k_1, k_2, \dots, k_t) \leftarrow \text{TrusteeKeys}(\omega_1, \dots, \omega_t, \tau, \lambda)$.
4. Generate election key: assuming the trusted platform is stateful during this phase, the election master key K is generated by the previous step. (Note that key K must not leave or be leaked from the trusted platform during computation, nor should the trusted platform be stateful between the setup, result declaration, and audit response phases. A minimum of τ keys from $\{k_1, k_2, \dots, k_t\}$ can regenerate all the information required for the result declaration and audit response phases.)
5. Generate ballots: the trusted platform computes $\mathbf{P} = \{\mathbf{Q}, \mathbf{R}, \mathbf{S}\} \leftarrow \text{GenerateBallots}(L, \Sigma, l, K, b)$, and transmits \mathbf{P} via a private channel to a trusted printing service which produces paper ballots with corresponding serial numbers and confirmation codes in invisible ink. Note that initially the recorded voter marks table \mathbf{R} is empty.
6. Shuffle P and cryptographically commit to the shuffles:

The following mixnet-like construction shuffles the two lists \mathbf{Q} and \mathbf{S} and posts commitments to the two shuffled lists and to the shuffles. The shuffles are constructed in a manner that will make the tally-verification audit simple to implement, as will be seen later. See Figure 4.3 for an illustration on the example of Figure 4.2. Note that, *in this example only*, we use cyclic permutations and a swap *merely* in an attempt to illustrate the mixnet-like construction in as simple a manner as possible. We *do not* advocate the restriction of permutations to a set of a few permutations, but, as mentioned below, require that each permutation be chosen in a pseudo-random manner from the set of all possible permutations of the respective tables.

- (a) Generate permutations: For each back-end, the trusted platform computes three permutations. That is, for $0 \leq i < I$, the trusted platform computes:
 - $\pi_{(i,1)} \leftarrow \text{Perm}(\text{Subkey}(K, \{\text{"1st"}, i\}), (b \cdot n))$
 - $\pi_{(i,2)} \leftarrow \text{Perm}(\text{Subkey}(K, \{\text{"2nd"}, i\}), (b \cdot n))$
 - $\pi_{(i,3)} \leftarrow \text{Perm}(\text{Subkey}(K, \{\text{"3rd"}, i\}), (b \cdot n))$
- (b) Shuffle lists: For each back-end the trusted platform computes a single-shuffled instance \mathbf{Q}'_i of \mathbf{Q} and a triple-shuffled instance \mathbf{S}'''_i of \mathbf{S} . Note the number of primes (') denote the number of shuffles that have been applied to the list list. That is, for $0 \leq i < I$ and $0 \leq j < (b \cdot n)$, the trusted platform computes:
 - $\mathbf{Q}'_i(j) \leftarrow \mathbf{Q}(\pi_{(i,1)}(j))$
 - $\mathbf{S}'''_i(j) \leftarrow \mathbf{S}_i(\pi_{(i,3)}(\pi_{(i,2)}(\pi_{(i,1)}(j))))$

j	α	β	γ	q
0	0000	7973	4630	WT9
1	0001	2567	1490	J3K
2	0001	2567	1490	TC3
3	0002	4900	7891	9JH
4	0002	4900	7891	J3K
5	0003	1631	5275	KWK
6	0003	1631	5275	H7T
7	0000	7973	4630	7LH

(a) Table \mathbf{Q}'_1

j	s
0	Bob
1	Alice
2	Bob
3	Alice
4	Bob
5	Alice
6	Alice
7	Bob

(b) Table \mathbf{S}'''_1

Figure 4.3: Scantegrity II Switchboard showing tables \mathbf{Q}' and \mathbf{S}''' for the example of Figure 4.2. \mathbf{Q}' is \mathbf{Q} permuted by $\pi_{(1,1)}$, which is an upward circular shift of one unit, and \mathbf{S}''' is \mathbf{S} permuted by $\pi_{(1,3)} \circ \pi_{(1,2)} \circ \pi_{(1,1)}$, where $\pi_{(1,2)}$ corresponds to an upward circular shift of two units, and $\pi_{(1,3)}$ swaps the last two elements in the list. Note that the confirmation numbers of \mathbf{Q}' can be made to match up with the correct candidates in \mathbf{S}''' if the permutation $\pi_{(1,3)} \circ \pi_{(1,2)}$ is applied to \mathbf{Q}' . Note that the confirmation codes do not appear publicly at this stage, rather cryptographic commitments to each code are posted instead. Note also that we use simple permutations such as these merely for the purposes of illustration. For the system to properly hide the association between codes and candidates, we require that each permutation be chosen (pseudo-randomly) from the set of **all** possible permutations.

- (c) Commitments: The trusted platform commits to each back-end—the shuffled confirmation code numbers, the corresponding candidate lists, and the permutation values—on an element-by-element basis. For each single-shuffled code list $\mathbf{Q}'_i(j) = \{\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}, q_{i,j}\}$, triple-shuffled candidate list $\mathbf{S}'''_i(j) = s_{i,j}$, and the corresponding elements of permutations $\pi_{(i,h)}(j)$ the trusted platform computes commitments as follows.

For $1 \leq h \leq 3$, $0 \leq i < I$ and $0 \leq j < (b \cdot n)$,

- $\bar{\alpha}_{i,j} \leftarrow \text{Commit}(\text{Subkey}(K, \{\text{“}\alpha\text{”}, i, j\}), \alpha_{i,j})$
- $\bar{\beta}_{i,j} \leftarrow \text{Commit}(\text{Subkey}(K, \{\text{“}\beta\text{”}, i, j\}), \beta_{i,j})$
- $\bar{\gamma}_{i,j} \leftarrow \text{Commit}(\text{Subkey}(K, \{\text{“}\gamma\text{”}, i, j\}), \gamma_{i,j})$
- $\bar{q}_{i,j} \leftarrow \text{Commit}(\text{Subkey}(K, \{\text{“}q\text{”}, i, j\}), q_{i,j})$
- $\bar{s}_{i,j} \leftarrow \text{Commit}(\text{Subkey}(K, \{\text{“}s\text{”}, i, j\}), s_{i,j})$
- $\bar{\mathbf{Q}}'_i(j) \leftarrow \{\bar{\alpha}_{i,j}, \bar{\beta}_{i,j}, \bar{\gamma}_{i,j}, \bar{q}_{i,j}\}$
- $\bar{\mathbf{S}}'''_i(j) \leftarrow \bar{s}_{i,j}$
- $\bar{\pi}_{(i,h)}(j) \leftarrow \text{Commit}(\text{Subkey}(K, \{\text{“}\pi\text{”}, h, i, j\}), \pi_{(i,h)}(j))$

- (d) The trusted platform publishes all $\bar{\mathbf{Q}}'_i$, $\bar{\mathbf{S}}'''_i$, and $\bar{\pi}_{(i,h)}$ to \mathcal{BB} .

7. The trusted platform’s internal state is purged.

A diagram of the Switchboard tables and permutations is shown in Figure 4.4.

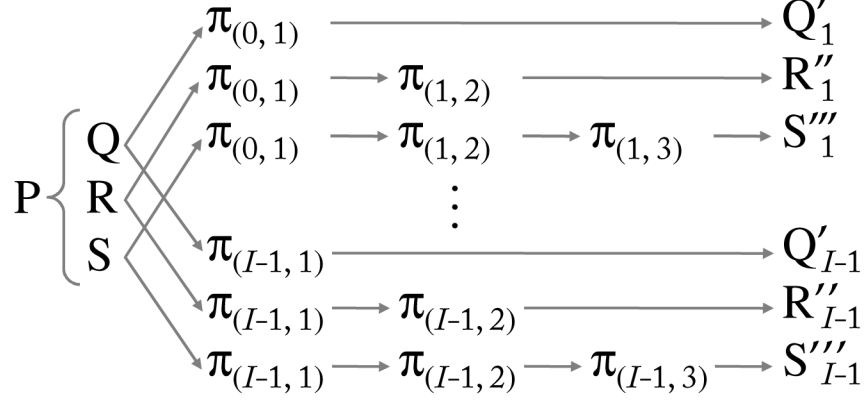


Figure 4.4: Topology of the Scantegrity II Switchboard. Master table \mathbf{P} links serial numbers (\mathbf{Q}) with voter marks (\mathbf{R}) for candidates (\mathbf{S}). \mathbf{P} is obfuscated by applying three independent random shuffles $\pi_{(i,1)}$, $\pi_{(i,2)}$, and $\pi_{(i,3)}$ to each table, as shown above. I independent instances of these shuffled tables are created and cryptographically committed to. For each instance i , one of either $\pi_{(i,2)}$ or $\pi_{(i,3)}$, can be challenged by the public to be revealed. In this way the correctness of \mathbf{P} can be audited without directly revealing the links between elements in \mathbf{Q} , \mathbf{R} , and \mathbf{S} .

Voting Phase. A voter V , upon being successfully authenticated by poll workers, is given a ballot $B_g = \{\alpha_g, \beta_g, \gamma_g, q_{gn}, q_{gn+1} \dots, q_{gn+n-1}, s_0, s_1, \dots, s_{n-1}\}$ consisting of a serial number α_g printed in an optical-scan readable “barcode” and selection/candidate list s_0, \dots, s_{n-1} printed in normal ink. Serial numbers β_g, γ_g and the corresponding confirmation codes $q_{gn}, q_{gn+1}, \dots, q_{gn+n-1}$ are printed in invisible ink.

To vote, V marks the optical scan bubble beside the desired selection s_d using the decoder pen, which reveals the confirmation code q_{gn+d} .

Upon scanning ballot B_g , the optical scanner shall produce an “electronic ballot image” $EBI_g = \{\alpha_g, r_{gn}, r_{gn+1}, \dots, r_{gn+n-1}\}$ where $r_{gn+d} = 1$ if and only a darkened region (*i.e.*, a mark) was detected inside the optical scan bubble beside the d -th selection s_d . All other $\hat{d} \neq d$ shall register $r_{gn+\hat{d}} = 0$. The specific electoral system in use would dictate how many marks (*i.e.*, distinct d 's) are permissible on a single ballot. V can then choose to construct a vote receipt $VR_g = \{\beta_g, \gamma_g, q_{gn+d}\}$ for each s_d marked.

Instead of voting on a particular ballot, V can select it to be “print-audited” in accordance with the procedures specified in Section 4.2. All confirmation codes are revealed, and **one of** $\{\beta_g, \gamma_g\} = \delta_g$ is revealed. The print-audited ballot becomes $PA_g =$

$\{\delta_g, q_{gn}, q_{gn+1}, \dots, q_{gn+n-1}, s_0, s_1, \dots, s_{n-1}\}$. For example, if Ballot 0001 of Figure 4.2 were not voted but were print-audited, and β_1 revealed, the print-audited ballot would be $PA_{0001} = \{2567, J3K, TC3, Alice, Bob\}$. (Note that print-audited ballots cannot be voted).

Declaring Results. After the polling concludes, a valid subset of trustees (as defined by τ) assemble to tally and declare the results. The trustees also make available data regarding the tally processing that will be audited in the audit phase. Given the set of all EBI s recorded during the election, the trustees proceed using the trusted platform as follows,

1. Regenerate election master key: Each trustee T_i transmits their trustee key k_i over an untappable channel to the trusted platform. The election master key K is reconstructed by calling $\text{ElectionKey}(\{j_1, \dots, j_n\})$ if at least τ trustees supply correct keys (where $\{j_1, j_2, \dots, j_n\}$ are n keys provided by n trustees).
2. Regenerate ballot list: Ballot list \mathbf{P} is reconstructed by rerunning step 5) of the setup phase,
3. Construct list of recorded marks: For each EBI_g recorded during the election, populate \mathbf{R} by setting $\mathbf{R}(gn + j) = r_{gn+j}$ for $0 \leq j < n$ and $r_{gn+j} \in EBI_g$. Any unused, spoiled or print-audited ballot B_g inherently constitutes an EBI_g with all $r_{gn+j} = 0$.
4. Post voted codes: During the dispute resolution period (described in Section 4.2) all voted codes shall be published. For all $\mathbf{R}(gn + d) = 1$ post $\{\beta_{gn+d}, \gamma_{gn+d}, q_{gn+d}\}$ and corresponding commitment keys.
5. Post results: Using \mathbf{P} tabulate the election results and post them to \mathcal{BB} ,
6. Post double-shuffled marks list for audit purposes:
 - (a) Regenerate permutations: For $0 \leq i < I$ and $0 \leq j < (b \cdot n)$ the trusted platform recomputes permutations:
 - $\pi_{(i,1)} \leftarrow \text{Perm}(\text{Subkey}(K, \{\text{"1st"}, i\}), (n \cdot b))$
 - $\pi_{(i,2)} \leftarrow \text{Perm}(\text{Subkey}(K, \{\text{"2nd"}, i\}), (n \cdot b))$
 - (b) Shuffle Lists: For $0 \leq i < I$ and $0 \leq j < (b \cdot n)$ the trusted platform computes I independent double-shuffled instances \mathbf{R}_i'' of \mathbf{R}_i
 - $\mathbf{R}_i''(j) \leftarrow \mathbf{R}(\pi_{(i,2)}(\pi_{(i,1)}(j)))$
 - (c) The trusted platform publishes all \mathbf{R}_i'' to \mathcal{BB} and purges its internal state.

α	β	γ	q
0000	7973	4630	WT9
0001	2567	1490	TC3
0002	4900	7891	9JH
0003	1631	5275	H7T

(a) Revealed values of Table \mathbf{Q}

j	r
0	1
1	1
2	0
3	0
4	1
5	0
6	1
7	0

(b) Table \mathbf{R}_1''

Figure 4.5: Example Switchboard Tables (Post-election). The revealed confirmation numbers, (entries in \mathbf{Q}) and revealed table \mathbf{R}_1'' , which is a shuffled version of \mathbf{R} , using the permutation $\pi_{(1,2)} \circ \pi_{(1,1)}$, where $\pi_{(1,1)}$ is an upward circular shift of one unit, and $\pi_{(1,2)}$ corresponds to an upward circular shift of two units. The tally will be “3 votes for Alice and one vote for Bob”. The permutations used are secret. Note that, if \mathbf{R}_1'' is permuted by $\pi_{(1,3)}$, the votes will be listed wrt candidate list \mathbf{S}''' of Figure 4.3. If list \mathbf{Q}'_i of the same figure is permuted by $\pi_{(1,2)}$, the confirmation codes will be listed as corresponding to the choices of \mathbf{R}_1'' above. Note also that we use simple permutations for the purposes of illustration. For the system itself, we advocate that each permutation be chosen pseudo-randomly from the set of all possible permutations, without restricting this set to the set of simple permutations such as cyclic permutations or swaps.

See Figure 4.5 for an illustration using the example of Figures 4.2 and 4.3.

Note that, at this stage, the list \mathbf{R}_i'' is such that, if permuted by $\pi_{(i,3)}$, the votes will be listed as obtained for the candidate list \mathbf{S}_i''' . Further, if list \mathbf{Q}_i' is permuted by $\pi_{(i,2)}$, the confirmation codes will be listed in the order of the votes \mathbf{R}_i'' .

Audit Challenge and Response. In order to ensure robust, correct behavior by the trustees and in turn, the correctness of the election outcome, two audits are carried out. We first describe the tally computation audit. For each back-end committed to by the trustee, a coin flip determines whether the trustees will demonstrate that the ballot marks of the corresponding public table \mathbf{R}_i'' correspond correctly to (a) the announced tally or (b) the public confirmation codes for voted ballots. This is done by opening the commitments to the permutation $\pi_{(i,3)}(j) \forall j$ or to the permutation $\pi_{(i,2)}(j) \forall j$ respectively. Second, we describe the print audit. For values of j , in the original ballot list \mathbf{Q} , corresponding to print-audited ballots, permutation values $\pi_{(i,2)}(j)$ and $\pi_{(i,3)}(j)$ are opened $\forall i$. We now describe these audits in more detail.

1. Public challenge of trustees: some time after the trustees have completed declaring the results and posting the shuffled marks lists, each instance of $\mathbf{Q}_i', \mathbf{R}_i'', \mathbf{S}_i'''$ is challenged to be partially revealed for the purposes of auditing. A fair public coin \mathcal{C} is tossed I times providing a series of audit challenges $\mathcal{C} \in \{0, 1\}^I$, which are posted to \mathcal{BB} .
2. For the tally computation audit. For $0 \leq i < I$ and $0 \leq j < (b \cdot n)$ the trusted platform performs the following actions:
 - (a) If $\mathcal{C}(i) = 0$, regenerate and publish the confirmation codes \mathbf{Q}_i' and the association between \mathbf{Q}_i' and \mathbf{R}_i'' . That is, regenerate and publish the following:
 - The second permutation $\pi_{(i,2)}$
 - The commitment subkeys of $\pi_{(i,2)} : \kappa_{\pi_{(i,2)}}(j) \leftarrow \text{Subkey}(K, \{\text{"}\pi\text{"}, 2, i, j\}) \forall j$
 - The commitment subkeys to all elements of $\mathbf{Q}_i' : \kappa_{x_{i,j}} \leftarrow \text{Subkey}(K, \{\text{"}x\text{"}, i, j\})$ where $x = \{\alpha, \beta, \gamma, q\} \forall j$
 - (b) If $\mathcal{C}(i) = 1$, regenerate and publish the permuted candidate list \mathbf{S}_i''' , as well as the association between \mathbf{R}_i'' and \mathbf{S}_i''' . That is, regenerate and publish the following:
 - The third permutation $\pi_{(i,3)}$
 - The commitment subkeys of $\pi_{(i,3)} : \kappa_{\pi_{(i,3)}}(j) \leftarrow \text{Subkey}(K, \{\text{"}\pi\text{"}, 3, i, j\}) \forall j$
 - The commitment subkeys to $\mathbf{S}_i''' : \kappa_{s_{i,j}} \leftarrow \text{Subkey}(K, \{\text{"}s\text{"}, i, j\}) \forall j$

3. For the ballot audit, compute all permutation elements and commitment keys not computed in tally audit and required for the purposes of demonstrating the entire path of the ballot through the mixnet-like construction. That is, the trusted computing platform does the following for $0 \leq i < I$

(a) If $\mathcal{C}(i) = 0$:

- Regenerate $\pi_{(i,3)}$ (do not publish it)
- For each ballot $PA_g = \{\delta_g, q_{gn}, q_{gn+1}, \dots, q_{gn+n-1}, s_0, s_1 \dots, s_{n-1}\}$ that is print-audited:
 - i. Search for all elements in \mathbf{Q}'_i such that the second component is δ_g . If there is no such element, search for all elements in \mathbf{Q}'_i such that the third component is δ_g . That is, find all j' such that $\mathbf{Q}'_i(j') = \{*, \delta_g, *, *\}$, failing which, find all j' such that $\mathbf{Q}'_i(j') = \{*, *, \delta_g, *\}$. For all such j' :
 - Compute $j''' \leftarrow \pi(i, 3)(j'')$ where $j'' \leftarrow \pi(i, 2)(j')$ has already been computed in the tally computation audit,
 - Publish $\pi(i, 3)(j'')$. Compute and publish the subkey used to commit to $\pi(i, 3)(j'')$: $\kappa_{\pi(i, 3)}(j'') \leftarrow \text{Subkey}(K, \{\text{"}\pi\text{"}, 3, i, j''\})$,
 - Publish $\mathbf{S}'_i(j''')$. Compute and publish the commitment subkey for this value: $\kappa_{s_{i, j'''}} \leftarrow \text{Subkey}(K, \{\text{"}s\text{"}, i, j'''\})$.

(b) If $\mathcal{C}(i) = 1$:

- Regenerate $\pi_{(i,2)}$ (do not publish it)
- For each ballot $PA_g = \{\delta_g, q_{gn}, q_{gn+1}, \dots, q_{gn+n-1}, s_0, s_1 \dots, s_{n-1}\}$ that is print-audited:
 - i. Search for all elements in \mathbf{Q}'_i such that the second component is δ_g . If there is no such element, search for all elements in \mathbf{Q}'_i such that the third component is δ_g . That is, find all j' such that $\mathbf{Q}'_i(j') = \{*, \delta_g, *, *\}$, failing which, find all j' such that $\mathbf{Q}'_i(j') = \{*, *, \delta_g, *\}$. For all such j' :
 - Compute $j'' \leftarrow \pi(i, 2)(j')$. Note that $j''' \leftarrow \pi(i, 3)(j'')$ has already been computed in the tally computation audit,
 - Publish $\pi(i, 2)(j')$. Compute and publish the subkey used to commit to $\pi(i, 2)(j')$: $\kappa_{\pi(i, 2)}(j') \leftarrow \text{Subkey}(K, \{\text{"}\pi\text{"}, 2, i, j'\})$,
 - Publish $\mathbf{Q}'_i(j')$. Compute and publish the commitment subkey for this value: $\kappa_{x_{i, j'}} \leftarrow \text{Subkey}(K, \{\text{"}x\text{"}, i, j'\})$ where $x = \{\alpha_g, \beta_g, \gamma_g, q\}$.

4.3.6 Correctness Proofs

We summarize the proofs of correctness that verifying agents A can perform and explicitly state conditions under which the proof completes successfully. Note that in general the

best practice response to proofs that do not complete successfully (*i.e.*, fail) is an open policy question, and not considered here. Specifically in case of voter receipts however, a failed receipt check has a dispute resolution process described in section 4.2.3.

Note that, for the print audit and tally check correctness proofs, A will verify commitments. In particular, A will confirm that all commitment keys that were challenged as a result of the challenge coin-tosses and the print-audit were responded to (*i.e.*, published on \mathcal{BB}) during steps 2 and 3 in the previous section. For all commitment keys κ_x to message x posted to \mathcal{BB} during the audit, A searches \mathcal{BB} for the corresponding message x and commitment value \bar{x} , and tests whether $\text{Decommit}(\kappa_x, x, \bar{x})$ outputs 1 (valid). This verification step is successful if and only if all of A 's executions of $\text{Decommit}()$ output 1.

Receipt Check. For all challenges $\mathcal{C}(i) = 0$ and $0 \leq j < (b \cdot n)$, A locates permutations $\pi_{(i,2)}$, code lists \mathbf{Q}'_i and recorded mark lists \mathbf{R}''_i on \mathcal{BB} . A reconstructs the assertion of the voting system, that $\mathbf{Q}'_i(j)$ is marked or not marked as indicated by the mark value $\mathbf{R}''_i(\pi_{(i,2)}(j))$.

This verification step successfully verifies voter-receipt $VR_g = \{\beta_g, \gamma_g, q_{gn+d}\}$ if and only if A is able to conclude that all reconstructed assertions agree with VR_g . Specifically for $0 \leq i < I$, VR_g is said to agree with the assertions if $\{\beta_g, \gamma_g, q_{gn+d}\}$ exists at position j in \mathbf{Q}'_i , if $\mathbf{R}''_i(\pi_{(i,2)}(j)) = 1$, and if all other occurrences of β_g and γ_g (that is, all $n - 1$ other values of tuples $\{\beta_g, \gamma_g, q_{gn+d}\}$ found at positions $\mathbf{Q}'_i(\hat{j})$ correspondingly show recorded mark $\mathbf{R}''_i(\pi_{(i,2)}(\hat{j})) = 0$.

Print Audit. A reconstructs assertions of the code-candidate associations of each print-audited ballot. For all i , $0 \leq i \leq I$, and for each print-audited ballot

$$PA_g = \{\delta_g, q_{gn}, q_{gn+1}, \dots, q_{gn+n-1}, s_0, s_1 \dots, s_{n-1}\} :$$

1. A searches for all elements in \mathbf{Q}'_i such that the second component is δ_g . If there is no such element, A searches for all elements in \mathbf{Q}'_i such that the third component is δ_g . That is, A finds all j' such that $\mathbf{Q}'_i(j') = \{*, \delta_g, *, *\}$, failing which, A finds all j' such that $\mathbf{Q}'_i(j') = \{*, *, \delta_g, *\}$. For all such j' :
 - A locates permutation element $\pi_{(i,2)}(j')$, computes $j'' \leftarrow \pi_{(i,2)}(j')$, locates permutation element $\pi_{(i,3)}(j'')$, computes $j''' \leftarrow \pi_{(i,3)}(j'')$,
 - A locates $\mathbf{R}''_i(j'')$ and $\mathbf{S}'''_i(j''')$,
 - The assertion is that $\mathbf{Q}'_i(j')$ is the confirmation number corresponding to the candidate $\mathbf{S}'''_i(j''')$ and that the unique ballot with one serial number δ_g has not been voted.

2. This verification step successfully verifies the print-audited ballot if it agrees with the assertions. That is, if A is able to obtain n values of j' and conclude that, $\forall j'$:

- The corresponding commitments were opened correctly,
- $\{\delta_g, q_{gn+\iota}\} \in \mathbf{Q}'_i(j')$ for some ι such that $0 \leq \iota \leq n - 1$ and that each value of ι corresponds to exactly one value of j' ,
- $\mathbf{R}''_i(j'') = 0$,
- $\mathbf{S}'''_i(j''') = s_\iota$.

Tally Check.

1. A will check that the corresponding commitments were opened correctly.
2. A will verify \mathcal{BB} self-consistency:
 - (a) For all $\mathcal{C}(i) = 0$, A performs the receipt check as described above. This verification step successfully verifies \mathcal{BB} self-consistency if all receipts verified by voters verify correctly.
 - (b) For all $\mathcal{C}(i) = 1$, A locates $\pi_{(i,3)}$, \mathbf{R}''_i and \mathbf{S}'''_i on \mathcal{BB} . For all j , A reconstructs the assertion of recorded mark $\mathbf{R}''_i(j)$ made for candidate $\mathbf{S}'''_i(\pi_{(i,3)}(j))$, and computes the election outcome by tallying each of these assertions. He checks the declared tally against the computed tally. This verification step successfully verifies \mathcal{BB} self-consistency if the two tallies are identical.

4.4 Security Analysis

In the previous section, we described the verification proofs for receipt checks, the tally check, and print-audited ballots. In this section, we both quantify the effectiveness of the verification and consider the security of the Scantegrity II to additional attacks, most involving a procedural element not easily captured by a cryptographic description. Thus, the goal of this analysis is to sketch the security heuristics underlying the design, and not to rigorously prove security properties in a formal cryptographic model.

We consider three categories of attacks. The first category are *manipulation attacks*, in which the goal of the attacker is to manipulate the final tally so that the election's outcome is more favorable to the attacker's preferred candidate(s). The second are *identification attacks*, where the goal of the attacker is to form a link between voting intent and ballot receipts. The final category are *disruption attacks*, in which the attacker wishes to prevent

the completion or certification of the election. Since, in general, disruption attacks are applicable in any voting system, and difficult to prevent, we will only consider a special-case of disruption involving the prevention of certification of any tally in the event the attacker feels the results may be unfavorable.

In order to best frame this discussion we note that, as an enhancement to optical scan, Scantegrity II is inherently constrained by our design goal of non-interference with the underlying optical scan processes. For this reason, Scantegrity II is designed to be a strict improvement over optical scan systems with manual recounts. However, components which cannot be secured without intervening in the underlying processes of optical scan are not pursued.

4.4.1 Assumptions

The level of security of Scantegrity II depends on the nature of the attack. Critical components offer probabilistic security that is invariant to the adversary's computational power, while other components premise their security on one or more assumptions, both procedural and cryptographic in nature. The security setting of our analysis includes the following assumptions,

1. The existence of a trusted computing platform for use by election officials (*contra* identification attacks),
2. The set of collusive officials in the election authority does not satisfy the threshold requirement for recovery of the master key (identification and disruption),
3. Chain-of-custody over the printed ballots prior to voting day (identification),
4. The inability of voters and others to read codes printed in invisible ink (manipulation, identification),
5. Proper balancing of the pollbook (manipulation),
6. The intractability of obtaining information about a message given only its cryptographic commitment (identification), and
7. The intractability of opening a cryptographic commitment of a message differing from that message initially committed to (manipulation).

In our view, most of these assumptions are reasonable and standard in the literature. The trusted platform is a scaled-down computing device, with no external memory, running

software attested by the trustees that performs the cryptographic operations. To avoid collusion among trustees, they could be selected from competing political parties. Using a threshold scheme allows the election to proceed even if a group of trustees is unable, or refuses, to supply their key share. Prior to the election, printed ballots must be protected against an adversary revealing codes and reprinting substitute ballots. Assumption 4 is unique to our approach and we provide justification for it in Section 4.5. “Balancing the pollbook” refers to the assumption that the sum of the number of voted, tallied and spoiled ballots is equal to the number of cast ballots, which is not larger than the number of voters. Assumptions 6 and 7 are referred to as the hiding and binding properties of commitments respectively in the previous section.

4.4.2 Manipulation Attacks

Printing. An adversary may misprint ballot B_g , so that the code $q_{gn+\hat{d}}$ associated with candidate $s_{\hat{d}}$ in the master list \mathbf{P} is printed beside a different candidate s_d (or all candidates) on the same ballot. If the adversary then modifies any EBI_g associated with such a misprinted ballot such that $r_{gn+\hat{d}} = 1$ and $r_{gn+d} = 0$, the system will count the vote for $s_{\hat{d}}$ and report q_{gn+d} as the confirmation code, which is consistent with what appears on the ballot.

The print audit mechanism, described in Section 4.3, is designed to make such an attack detectable by revealing discrepancies between printed ballots and \mathbf{Q}'_i , using commitments $\bar{\mathbf{Q}}'_i$ under assumption 7. If the number of ballots chosen to be print-audited is $0 \leq b_a \leq b$ where b is the number of ballots in the election overall, the probability of detecting at least 1 of $1 \leq b_f \leq b$ misprinted ballots is,

$$\begin{aligned} \Pr[\text{detection}] &= 1 - \frac{\binom{b-b_f}{b_a}}{\binom{b}{b_a}} \\ &= \frac{(b-b_f)!(b-b_a)!}{b!(b-b_f-b_a)!} \end{aligned} \tag{4.1}$$

Voting. One line of manipulation attack can exist in systems that are not diligent in spoiling ballots [KRMC07, Ben07]. If an attacker has a line of communication with the voter, the voter can be instructed to mark her ballot and wait for further instruction. The attacker then communicates to the voter to either spoil the ballot or cast it. If the spoiled ballot is not protected or destroyed, the attacker may consult it to see how the voter would have voted had the attacker instructed the voter to cast the ballot. The line of communication can be eliminated by using random material on the ballots to determine

the instruction, in a way analogous to the approach of making interactive protocols non-interactive. Scantegrity II avoids this line of attack by having spoiled ballots shredded in front of the voter, without the poll worker seeing the contents of the ballot.

A second line of manipulation attack can exploit the presence of undervoted ballots. An attacker may add additional marks to a contest left empty by the voter during a recount or appropriately modify the digital records.² This attack is not introduced by Scantegrity II and exists in any optical scan voting system. One method of prevention is to require each voter to mark a “none of the above” selection when denoting an undervote. Similarly, an attacker might try to prevent a correctly-cast ballot from being tallied by overvoting it; this attack is prevented by not allowing any overvoted ballots to be cast.

Auditing. Consider a manipulation attack based on swapping voter-made marks in \mathbf{R}_i'' from one candidate to the attacker’s preferred candidate. To prevent this attack, with probability $\frac{1}{2}$, each back-end will be challenged to open the correspondence between the lists \mathbf{R}_i'' and \mathbf{Q}_i' , and any modified mark states for these instances will be incongruent with the voter receipts. The attacker may gamble, only modifying marks in roughly half of the back-end instances in the hope that exactly these will have challenge $\mathcal{C}(i) = 1$ and thus that, instead, that the correspondence between the lists \mathbf{R}_i'' and \mathbf{S}_i'' is instead revealed in the modified instances i . The probability of doing so is 2^{-I} . However if a different subset is revealed, the tallies across the subsets will differ and the attack is detectable. Alternatively, the attacker might modify \mathbf{R}_i'' for all instances $1 \leq i \leq I$, which guarantees self-consistent tallies but also guarantees the attack is detectable by the receipt check protocol. At first glance this may seem to be an irrational strategy until one considers the possibility of only a small subset of voters actually checking their receipts. With I instances, $b_r \leq b$ ballots actually cast, b_c ballot receipts checked, and b_m modifications to each \mathbf{R}_i'' , the probability of detection is $(b_r - b_m)!(b_r - b_c)!/b_r!(b_r - b_m - b_c)!$. The adversary will choose the least detectable of the two strategies, thus,

$$\Pr[\text{det.}] = \min\left(1 - \frac{1}{2^I}, 1 - \frac{(b_r - b_m)!(b_r - b_c)!}{b_r!(b_r - b_m - b_c)!}\right). \quad (4.2)$$

By estimating b_c and bounding b_m as half of the smallest margin of victory we can certify an election for, we can use this equation to determine a suitable I for our implementation such that the first term exceeds the estimated value of the second. In most instances, $I = 10$ is suitable.

²Although this issue was previously known to the authors, we acknowledge David Wagner for raising it in private correspondence.

A second approach to manipulating the tally is to change the final state of the ballots. Ballots can have one of three states: voted, print-audited, or spoiled. Under assumption 5, we assume that modifications must preserve the number of ballots in each state. If a voted ballot is maliciously modified to be spoiled, a spoiled ballot must be converted into a voted ballot. To prevent these transitions, the voter retains positive evidence of ballots being in a voted state: knowledge of both serial numbers, $\{\beta_g, \gamma_g\}$. Alternatively, for a print-audited ballot, the voter retains positive evidence a ballot was print-audited via knowledge of all the confirmation codes on the ballot, $\{q_{gn}, q_{gn+1} \dots q_{gn+n-1}\}$ but only *one* of $\{\beta_g, \gamma_g\}$. Both pieces of information would be unknown to the voter if the ballot were in any other state when the voter left the polling place.

In the case of spoiled ballots, the voter does not retain anything. However, if a spoiled ballot is maliciously converted into a voted ballot, a voted ballot will need to be spoiled, and the corresponding voter can prove malfeasance through knowledge of both chit serial numbers. Of course an adversary could attempt to choose a voter who appears unlikely to check his or her receipt, and is a practical limitation of our approach.

The transition from a spoiled to print-audited state is important for different reasons. This transition does not change the tally directly, however it is indirectly useful in facilitating the first manipulation attack presented in Section 4.4.2. By misreporting a spoiled ballot as print-audited, the confirmation codes on the ballot would be released during the verification process allowing a coercer to see if the ballot matched the conditions of the contract for spoiling the ballot. Under assumption 5, this attack will be detectable as it requires a print-audited ballot to be made into a spoiled ballot. To prevent this attack, the trustees could first publish a list of ostensibly spoiled ballots prior to releasing the print audit confirmation codes. If an auditor discovers her print-audited ballot is in the wrong state, the discrepancy can be caught prior to releasing the codes.

4.4.3 Identification Attacks

Initialization. The earliest opportunity for identification occurs during the election initialization process. Successfully changing or introducing faults into the initialization protocol could generate a permutation of \mathbf{P} or subsequent lists that is known to the attacker. This is not possible if the protocol is run on a trusted computing platform and assumption 1 holds. Without direct interference with the protocol, the attacker may provide structured data instead of randomness in the protocol. However under assumption 2 and the construction of the threshold key generation scheme, any amount less than the minimum threshold of shares leaks negligible information for the purposes of determining the key.

Printing. After the ballots are printed, a number of identification attacks may be conducted including the addition of revealing marks on the ballots or revealing the codes on

the ballots, recording these codes, and reprinting the ballots with unrevealed ink. The prevention of these attacks is based on assumption 3.

Auditing. After the election has concluded, the data generated and published for voter-verification of the tally must meet the requisite ballot secrecy. Given no information other than the tally, a certain level of information can be obtained about which candidate a voter selected. The tally provides a probability distribution for the possible selections and may even exclude selections, based, for example, on a candidate receiving zero votes. This level of information is often legally required and thus acceptable. If the attacker is provided, in addition, with the information on each voter’s receipt, further information is revealed: how many marks the voter made and the codes associated with these marks. Our assertion of ballot secrecy is that no additional information is leaked about the association between a mark and code on a receipt and any element in the set of selections in the tally.

Opening only one of the commitments to either (a) the correspondence between confirmation codes \mathbf{Q}'_i and voter marks \mathbf{R}''_i or (b) the correspondence between voter marks \mathbf{R}''_i and candidates \mathbf{S}'''_i reveals no information about permutations $\pi_{(i,3)}$ or $\pi_{(i,2)}$ respectively. Hence the association between \mathbf{Q} and \mathbf{S} is always hidden by one cryptographic permutation. The commitment to the permutation key, if binding, uniquely identifies the permutation however reversing the commitment is assumed intractable by assumption 6.

4.4.4 Disruption Attacks

In general, disruption attacks are easy to detect but difficult to prevent. Many of the manipulation attacks could be reconstructed as disruption attacks, and the same mechanisms would detect them. However, as stated, we limit our consideration to disruption for the purpose of preventing the certification of an undesired tally (or an expected undesired tally, if the information is based on exit polls for example).

Initialization. During the initialization phase, each trustee in the election authority supplies entropy to seed the random number generator used to generate all the permutation keys and commitment secrets needed in the election. Instead of maintaining state, since the state information would need to remain private, when the tally and audit challenge/response phases are entered, the trustees re-enter their key shares to recreate all the necessary data. To prevent a malicious trustee from withholding their entropy or supplying the wrong entropy, we use a threshold key generation scheme (optionally with robustness to a finite number of errors). Under assumption 2, a suitable threshold will allow the reconstruction of the data despite malicious trustees.

Auditing. During the auditing phase, an attacker may file a spurious dispute about the results of a receipt-check. Since the election authority has committed to the confirmation codes that appeared on the ballot, it can rule out any claimed codes that did appear on the ballot. Thus, filing a spurious but plausible dispute reduces to randomly guessing another code on the ballot. The election authority can quantify the probability of this and create an appropriate statistical trigger that predicts actual receipt-check problems. Let n be the number of candidates on a candidate list L for a particular race and let Σ^l be the cardinality of the set of unique confirmation codes. The probability of guessing a plausible code on a voted ballot is $p = (n - 1)/(\Sigma^l - 1)$. If D disputes are filed and G are considered plausible, the expected value of G if disputes are fabricated is $\mu = D \cdot p$. We set the trigger value $\tau\nu$ such that the probability of obtaining at least $\tau\nu$ plausible discrepancies if all filed disputes are random guesses is less than some acceptably small amount (e.g., $> 1\%$). We can use the following bound on the right tail of the binomial distribution [CLRS00]. For any $r > \mu$, $\Pr[G - \mu \geq r] \leq (\mu e/r)^r$.

For example, for 5 candidates, 8000 possible codes, and 1000 disputes filed, assuming no scanning error, $p = 4/7999 = 0.0005$ and $\mu = 1000 \cdot 0.0005 = 0.5$. Using $r = 4.5$ we get $\Pr[G \geq 5] \leq (0.5e/4.5)^{4.5} = 0.0046 < 0.01$, so we can set $\tau\nu = 5$. If at least 5 out of the 1000 disputes filed are plausible discrepancies, then an investigation should be instigated. To allow for up to some acceptable rate s of scanning error, we can incorporate s into the probability p of guessing a correct code and compute the statistical trigger as above with the new value of p .

4.5 Invisible Ink

Recalling our description of invisible ink from Section 2.1.6 we define a matrix of pixels for the purposes of printing the confirmation codes inside the optical scan bubbles. Pixels that form the “foreground” confirmation code characters are printed in one ink, while the remaining “background” pixels are printed in the other ink. In order to produce the darkest average color density of a marked optical scan bubble (and therefore most visible to the scanner), the background pixels are printed with the invisible ink, and the foreground pixels are printed in the non-reactive dummy ink (or slower-reacting ink if the application warrants disappearing codes).

The developer ink is incorporated into a “decoder pen”, a felt-tipped highlighter style marker along with a basic yellow pigment to provide voters with visual feedback as to where they have marked. A photograph of the decoder pen activating the invisible ink in an optical scan oval is shown in Figure 4.6.

We implemented the invisible ink printing process using an Epson C88+ color inkjet printer in which we replace the manufacturer-supplied yellow and magenta inks with the

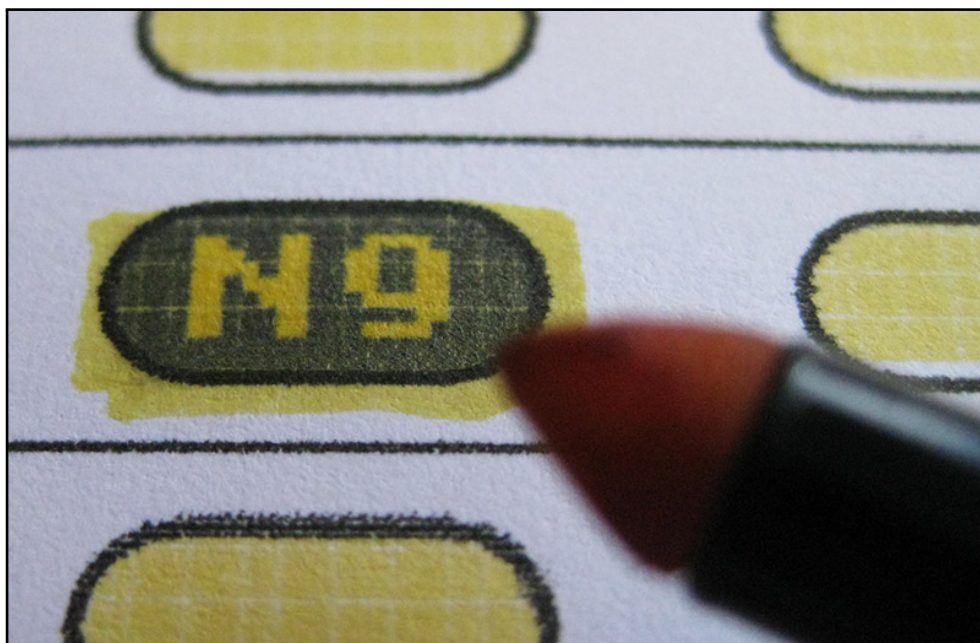


Figure 4.6: Decoder Pen Activating Invisible Ink.

reactive and non-reactive inks respectively. Electronic files used for ballot printing are prepared in this false-color mapping whereby yellow and magenta colored regions in the digital file will print in reactive and non-reactive inks respectively. The invisible ink printing process is depicted in Figure 4.7.

4.5.1 Invisible Ink Threat Model

Threats to Scantegrity II that take advantage of the limitations of the invisible ink are those in which an adversary is able to:

1. Distinguish between confirmation codes and their backgrounds. The ability to distinguish would allow:
 - (a) voters to falsely claim election fraud,
 - (b) anyone with access to ballots to violate ballot secrecy by connecting confirmation codes to selections,
2. Distinguish between chit serial numbers and backgrounds. The ability to distinguish would allow:
 - (a) voters to claim that an uncast ballot was cast,

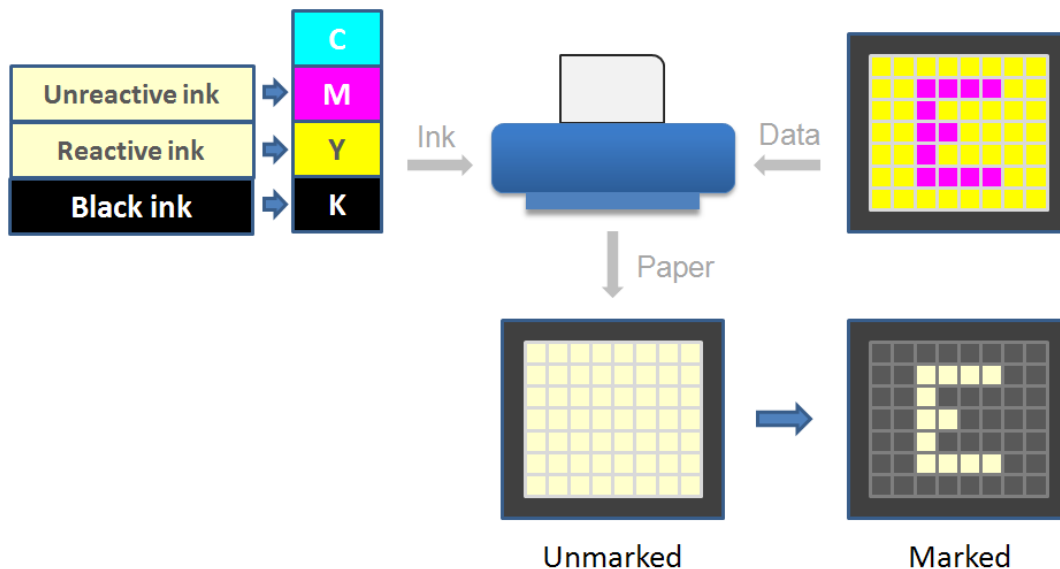


Figure 4.7: Invisible Ink Printing Process.

- (b) anyone with access to uncast ballots to connect chit serial numbers and confirmation codes with voter selections (in combination with (1)),
- 3. Distinguish between the two-dimensional barcode and background. The ability to distinguish would allow anyone with access to marked ballots to connect two-dimensional bar-codes with voter selections.

4.5.2 Invisible Ink Assumptions

The main security assumption about the inks is that the slow and fast reacting inks used for printing confirmation codes and oval backgrounds respectively are not distinguishable before, and sufficiently after, they have been marked with the ballot-marking pen (“sufficiently after” is taken to mean that the time period is long enough to allow the slow-reacting ink to react). We make a similar assumption about the indistinguishability of the chit serial numbers and the two-dimensional barcode from their background.

Note also that the assumptions we make are about physical properties of chemicals, and the detectability of differences. Clearly, most chemicals (if not all) can be distinguished from one another through a sufficiently sophisticated test; our arguments are that, for all practical purposes, our assumption holds, and we describe here our efforts to make it more difficult to distinguish among the inks, particularly by the naked or microscopically-aided human eye.

Finally, the ability to distinguish enables voters to make false charges of election fraud, and anyone to connect information about ballot choices with confirmation codes and serial numbers. If voters are assumed to not have access to ballots outside the polling booths, or to specialized equipment (including the decoding pen) inside the booth, the indistinguishability assumption is only required to hold with respect to the human eye in order to prevent false charges of election fraud.

4.5.3 Procedures For Printing With the Inks

In this section we describe ways in which the indistinguishability assumptions may be defeated, and our efforts to preserve indistinguishability. Note that the inks proposed for printing on ballots can be used in regular ink-jet printers.

To prevent the soaking of paper. Any type of ink used by inkjet printers soaks into the paper. Even if the ink used to print the codes would be completely invisible, the soaked paper would allow the codes to be easily read. To avoid this, we use two types of ink: a reacting ink used to print the background of the oval and a slow-reactive ink used to print the confirmation codes. Both inks have the same color (a light yellow) if printed on the same piece of paper. The reacting ink turns black immediately when it interacts with the ink of the marking pen, while the ink used for the codes undergoes the same reaction at a slower pace. Thus, the immediate result is a yellow confirmation code inside a black oval—the highest contrast color combination. After several minutes, the slow reacting ink will have reacted leaving the oval completely black.

To avoid the overlapping of inks. We divide an optical-scan oval in small square pixels. Each pixel is entirely printed with either reactive or slow-reactive ink, but never with a combination of them. A small constant-sized gap is left between any two adjacent pixels, such that when two adjacent tiles are printed with different inks, the two inks never overlap even if they diffuse outward as they absorb into the paper. Without such a gap, a border of overlapping types of ink could emerge, under a microscope, for example, making the border easier to detect. Additionally, we ensure that the position of the code in the oval is not fixed; the codes can be shifted left or right.

The addition of camouflaging fluorescence. The use of special types of radiation can expose invisible inks. Our initial experiments printing with invisible ink demonstrated that the confirmation codes became visible under fluorescent/UV light.

To safeguard the codes against passive attacks, we developed a UV camouflaging process using a UV-reactive “masking ink” that is colorless under normal lighting conditions but



Figure 4.8: UV-Reactive Invisible Ink Camouflaging.

has high fluorescence/reactivity under UV light. After the invisible/dummy inks have been applied, independent random amounts of masking ink is applied to each pixel of the oval. This is designed to mask the eventual difference in fluorescence between the reactive ink and slow reactive ink used for the codes, as well as a cover to prevent lifting particles from the paper with tape. A photograph depicting camouflaged invisible ink optical-scan ovals under UV light is shown in Figure 4.8.

Ballot-marking pens. The ballot-marking/“decoder” pens that we use to mark the ovals have a tip that is wider than the height of the oval. A voter can mark the entire oval using a single strike of the pen which is faster than penciling in the mark. Even if the voter pens in more than the oval, the result is a clean, perfectly filled oval. The use of invisible ink also deters stray dark marks that can confuse scanners, although the light yellow hue of the ink could still be visible. The portion of the chit reserved for the voter to record the confirmation codes can also have a solid layer of the same reacting ink, so that the voter may record the codes with the same pen.

4.6 Concluding Remarks

Like its predecessor Scantegrity, Scantegrity II provides voters with a familiar optical scan voting experience and an opt-in receipt creation procedure. Since receipt creation is unsupervised, disputes can occasionally arise between voters and election officials over which confirmation code correctly reflected what the voter saw in the booth. As a main contribution of this chapter, Scantegrity II introduced an informational dispute resolution

procedure that improves upon that of its predecessor by eliminating the requirements that the voter appear in-person, bearing the original receipt chit, and the associated physical/forensic requirements of authenticating it against the original cast ballot. In Scantegrity II, knowledge of the correct confirmation code suffices to resolve the dispute (in either direction), allowing the process to be automated and even conducted remotely.

Up until this point in our research, however, there were many unanswered questions of a practical nature: would voters be confused by the role of the invisible ink? How many would create and check their receipt? Would we be able to print Scantegrity II ballots *en masse*? Put simply: how would the system perform in reality? We will tackle these questions in the following chapter.

Chapter 5

Scantegrity in Practice at Takoma Park, MD

Cryptographic voting is definitely not ready for prime time.

David Dill [Dil07]

There are cryptographic techniques ... but those are not ready for prime time in my opinion.

Avi Rubin [Wei08]

This chapter is adapted from published work co-authored with David Chaum et al. [CCC⁺10].

5.1 Introductory Remarks

In this chapter we present a case study of Scantegrity II's use in the 2009 municipal election of Takoma Park, Maryland, beginning in February 2008 when the Scantegrity research team was approached by Takoma Park officials, to the final cryptographic audit in December 2009.

Despite several limitations of the implementation, we found that the amount of extra work needed by officials to use Scantegrity II while administering an election is acceptable given the promise of improved voter satisfaction and indisputability of the outcome.

Another observation from the election is that the election officials and voters surveyed seemed to appreciate the system. Since voters who do not wish to verify can simply proceed as usual, ignoring the codes revealed in the filled ovals, the system is least intrusive for these voters. Those voters who did check their codes, and even many who did not, seem to appreciate the opportunity.

We describe the process of developing Scantegrity II to handle the Takoma Park election specifications, which includes the original agreement with the city, printing the special ballots with invisible-ink confirmation codes, actually running the election, and verifying that the election outcome was correct. Throughout this chapter we interchangeably use the term “Scantegrity” either to refer to the Scantegrity II voting system, or the Scantegrity research team, which should be clear in context.

Contributions. The contributions of this chapter are summarized as follows:

- A case study of the first governmental election to use a paper-based cryptographically verifiable election system (with Scantegrity II),
 - The adaption of the basic Scantegrity II ballot and protocol to a Instant-runoff Vote (IRV) style election,
 - A full-scale implementation,
 - Results of an exit survey of voter impressions.

5.2 The Setting

For several reasons, the implementation of voting systems is a difficult task. Most voting system users—*i.e.*, the voters—are untrained and elections happen infrequently. Voter privacy requirements preclude the usual sorts of feedback and auditing methods common in other applications, such as banking. Also, government regulations and pre-existing norms in the conduct of elections are difficult to change. These issues can pose significant challenges when deploying new voting systems, and it is therefore useful to understand the setting in which the election took place.

5.2.1 About Takoma Park

The city of Takoma Park is located in Montgomery County, Maryland, shares a city line with Washington, D.C, and is governed by a mayor and a six-member City Council. The city has about 17,000 residents¹ and almost 11,000 registered voters. A seven-member Board of Elections conducts local elections in collaboration with the City Clerk. In the past, the city has used hand counts and optical scan voting, as well as DREs for state elections.

The Montgomery County US Census Update Data of 2005 provides some demographic information about the city. Median household income in 2004 was \$48,675. The percentage of households with computers was 87.4%, and about 32% of Takoma Park residents above the age of twenty-five had a graduate, professional or doctoral degree. It is an ethnically diverse city: 45.8% of its residents identify their race as “White,” 36.3% as “Black,” 9.7% as “Asian or Pacific Islander” and 8.2% as “Other” (individuals of Hispanic origin form the major component of this category). Further, 44.4% of its households have a foreign-born head of household or spouse, and 44.8% of residents above the age of five spoke a language other than English at home.

Instant Runoff Voting (IRV). Takoma Park has used IRV in municipal city elections since 2006. IRV is a ranked choice system where each voter assigns each candidate a rank according to her preferences. The rules² used by Takoma Park (and the Scantegrity software) for counting IRV ballots are relatively standard.³

5.2.2 Agreement with the City

As with any municipal government in the US, Takoma Park is allowed to choose its own voting system for city elections. For county, state, and federal elections, it is constrained by county, state, and federal election laws.

Takoma Park and the Scantegrity Voting System Team (SVST) signed a Memorandum of Understanding (MOU), in which the SVST agreed to provide equipment, software, training assistance, and technical support. The City of Takoma Park agreed to provide election-related information on the municipality, election workers, consumable materials, and perform or provide all other election duties or materials not provided by us. No goods or funds were exchanged.

¹See <http://www.takomaparkmd.gov/about.html>.

²For the exact laws used by Takoma Park, see page 22 of <http://www.takomaparkmd.gov/code/pdf/charter.pdf>. Section (f), concerning eliminating multiple candidates, was used in our implementation for tie-breaking only.

³See e.g., http://en.wikipedia.org/wiki/Instant_Runoff_Voting

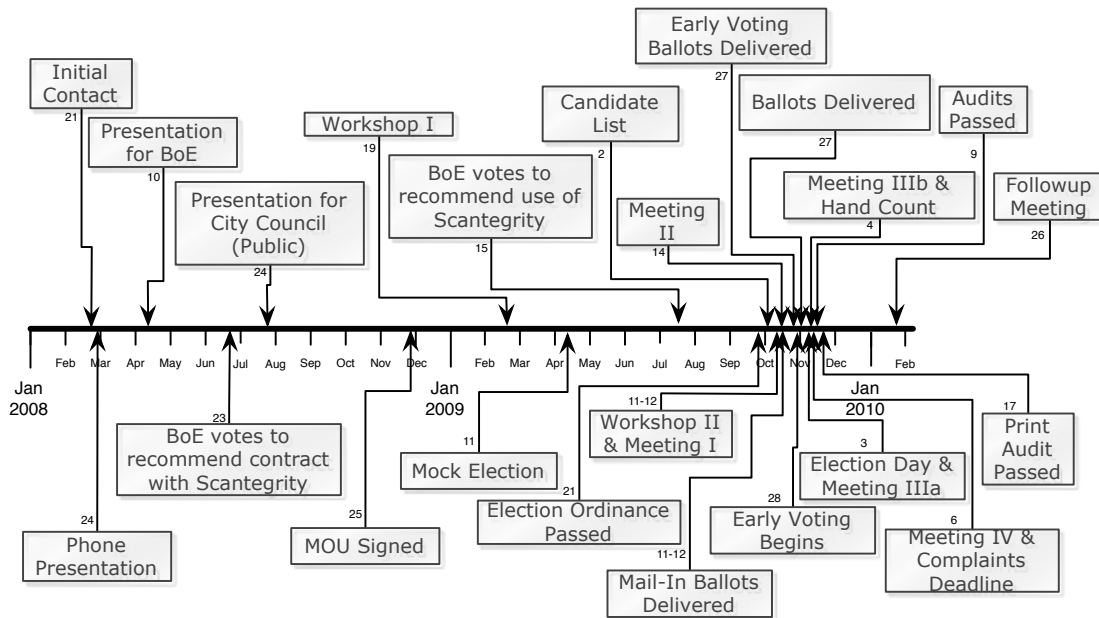


Figure 5.1: Timeline of 2009 Takoma Park Election. Each box represents an event, and the numbers next to each box are the day(s) the event took place during the indicated month.

According to the MOU, if approved by the city council, the election was to be conducted in compliance with all applicable laws and policies of the city. This included using Instant Runoff Voting as defined by the City of Takoma Park Municipal Charter.

The SVST also agreed to pursue an accessible ballot-marking device for the election, but was later relieved of satisfying this requirement. Unfortunately, Scantegrity is not yet fitted with a voter interface for those with visual or motor disabilities, and accessible user interfaces were also not used in Takoma Park’s previous optical scan elections.

5.2.3 Timeline

Scantegrity was approached by the Takoma Park Board of Elections in late February 2008, and, after considering other voting systems, the Board voted to recommend a contract with Scantegrity in June 2008. Following a public presentation to the City Council in July 2008, the MOU was signed in late November 2008, about nine months after the initial contact. As part of the Scantegrity agreement with Takoma Park, both parties would run a “mock election” in partnership as a feasibility test to ascertain the system’s readiness and to make any necessary changes or improvements for the actual election.

The SVST held an open workshop in February 2009 to plan and discuss the mock and

actual elections. This workshop was held at the Takoma Park Community Center and was attended by Board of Election members, the City Clerk, current members (and a retired member) from the Montgomery County Board of Elections, as well as a representative each from the Pew Trust and FairVote.

The mock election was held April 2009 during Takoma Park’s annual Arbor Day in which local residents cast ballots for their favorite tree. The BoE requested a number of revisions and tweaks be made to the Scantegrity system as a result of the mock election, including: ballot revisions (no detachable chit,⁴ but instead a separate voter verification card), pen revisions (two-ended, with different sized tips), scanner station revisions (better voter flow, no monitor, two scanners), privacy sleeve (no lock, no clipboard, folding design, feeds directly into scanner), and confirmation codes (three decimal digits). We also took the opportunity to conduct a user study of Scantegrity during the mock election [SCC⁺10], the results of which are outside the scope of this work.

Following the mock election, the SVST proposed a redesigned system taking into consideration feedback from voters and poll workers (through surveys) and the Board of Elections. The Board voted to recommend use of the redesigned system in July 2009; this was made official in the city election ordinance in September 2009.⁵ Beginning around June 2009, a meeting with representatives of the SVST was on the agenda of most monthly Board of Election meetings. Additionally, SVST members met many times with the City Clerk and the Chair of the Board of Elections to plan for the election.

The final list of candidates was available approximately a month before the election, on October 2. The Scantegrity meetings initializing the data and ballots were held in October (see Section 5.5), as was a final workshop to test the system. Absentee ballots were sent out by the City Clerk in the middle of October. The SVST delivered ballots to the City Clerk in late October, and early voting began almost a week before the election, on October 28. Poll worker training sessions were held by the city on October 28 and 31, and polling on November 3, 2009, from 7 am to 8 pm. The final Scantegrity audits were completed on 17 December 2010; all auditors were of the opinion that the election outcomes were correct (for details see section 5.5).

5.3 Scantegrity Overview

This section describes some of the real-world particulars of Scantegrity at Takoma Park.

⁴Unlike Scantegrity, the information dispute resolution procedure of Scantegrity II does not require a physical connection between ballot and receipt.

⁵See <http://www.takomaparkmd.gov/clerk/agenda/items/2009/090809-3.pdf>, section 2-D, page 2.

The Ballot. The Scantegrity ballot looks similar to a conventional optical scan ballot. See Figure 5.2 for a sample of the ballot used in the election. It contains a matrix of the choices and optical scan ovals allowing the voter to rank their choices. Marking an oval reveals a random 3-digit confirmation code.

Confirmation Codes. The confirmation codes are unique within each contest on each ballot, and are generated independently and uniformly pseudorandomly. The confirmation code corresponding to any given choice on any given ballot is hidden and unknown to any voter until the voter marks the bubble for that choice.

Voter Experience. To vote, a voter first checks in at the polling place and receives a Scantegrity ballot in a privacy sleeve. The privacy sleeve is used to cover the ballot and keep private the contents of the ballot. Inside the voting booth, there is a special “decoder pen” and a stack of blank “voter verification cards.” The voter uses the decoder pen to mark the ballot. As on a conventional optical scan ballot, she fills in the bubble next to each of her selections. Marking a bubble with the decoder pen simultaneously leaves a dark mark inside the bubble and reveals a previously hidden confirmation code printed in invisible ink. Additionally voters were required to rank their preferences as part of the instant-runoff voting (IRV) style employed by Takoma Park (see Figure 5.3(a) for a depiction of how this ballot style looks when marked).

If a voter makes a mistake, she can ask a poll worker to replace her ballot with a new one. The first ballot is marked “spoiled,” and its ballot ID is added to the list of spoiled ballot IDs maintained by the election judges.

Verification Card. If the voter wishes to verify her vote later on the election website, she can copy her ballot ID and her revealed confirmation codes onto a voter verification card. She keeps the verification card for future reference. She then takes her ballot to the scanning station and feeds the ballot into an optical scanner, which reads the ballot ID and the marked bubbles. The verification card is shown in Figure 5.3(b). Spanish instructions were printed on the reverse. Relevant fields were printed with reactive ink, allowing voters to use the same pen to mark the ballot and record confirmation numbers. Voters were issued a two sided pen with chisel- and bullet-tips.

The voter can verify her vote on the election website by checking that her revealed confirmation codes and ballot ID have been posted correctly. If she finds any discrepancy, the voter can file a complaint through the website, within a complaint period. When filing a complaint, the voter must provide the confirmation codes that were revealed on her ballot as evidence of the validity of the complaint.

Ward number → 1-392060
Stub Number:

**City of Takoma Park, Maryland
MUNICIPAL ELECTION
NOVEMBER 3, 2009**

OFFICIAL BALLOT — WARD 1

Instructions: Vote for candidates by indicating your first-choice candidate, your second-choice candidate, and so on. You are free to rank only a first choice if you wish.

Do not fill in more than one oval per column. Do not fill in more than one oval per candidate. Do not skip numbers in the ranking sequence.

To vote for a person whose name is not printed on the ballot, write the name in the space provided and fill in one box in the column indicating your ranking of the write-in candidate.

If you make a mistake on your ballot, return it to the judge and get another.

Do not make any identifying marks on your ballot.

When you mark an oval to rank a candidate, a code will be revealed that you may later use to verify your vote online. See the instruction sheet in the voting booth.

**Ciudad de Takoma Park, Maryland
ELECCIONES MUNICIPALES
3 DE NOVIEMBRE DE 2009**

BOLETA OFICIAL— DISTRITO ELECTORAL 1

Instrucciones: Vote por los candidatos indicando el candidato que sea su primera opción, el candidato que sea su segunda opción, y así sucesivamente. Si lo desea, puede limitarse a seleccionar solamente al candidato que sea su primera opción.

No rellene más de una casilla por cada columna. No rellene más de una casilla por cada candidato. No salte números en la secuencia de clasificación por orden.

Para votar por una persona cuyo nombre no esté impreso en la boleta, escriba el nombre en el espacio provisto y rellene una casilla en la columna para indicar el orden de clasificación del candidato que se ha añadido.

Si usted comete un error en su boleta, devuélvasela al juez y pida otra.

No haga marcas en su boleta que puedan identificarlo.

Cuando usted marque la casilla para votar por un candidato, verá un código que podrá usar posteriormente para verificar su voto por Internet. Vea la hoja de instrucciones en la cabina de votación.

MAYOR ALCALDE			
Rank candidates in order of choice <i>Clasifique a los candidatos por orden de preferencia</i>	1st choice <i>1ra opción</i>	2nd choice <i>2da opción</i>	3rd choice <i>3ra opción</i>
Roger B. Schlegel	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bruce Williams	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Write-In Candidate/ <i>Para añadir a un candidato</i>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

CITY COUNCIL MEMBER WARD 1 <i>MIEMBRO DEL CONSEJO DE LA CIUDAD DISTRITO ELECTORAL 1</i>		
Rank candidates in order of choice <i>Clasifique a los candidatos por orden de preferencia</i>	1st choice <i>1ra opción</i>	2nd choice <i>2da opción</i>
Josh Wright	<input type="radio"/>	<input type="radio"/>
Write-In Candidate/ <i>Para añadir a un candidato</i>	<input type="radio"/>	<input type="radio"/>

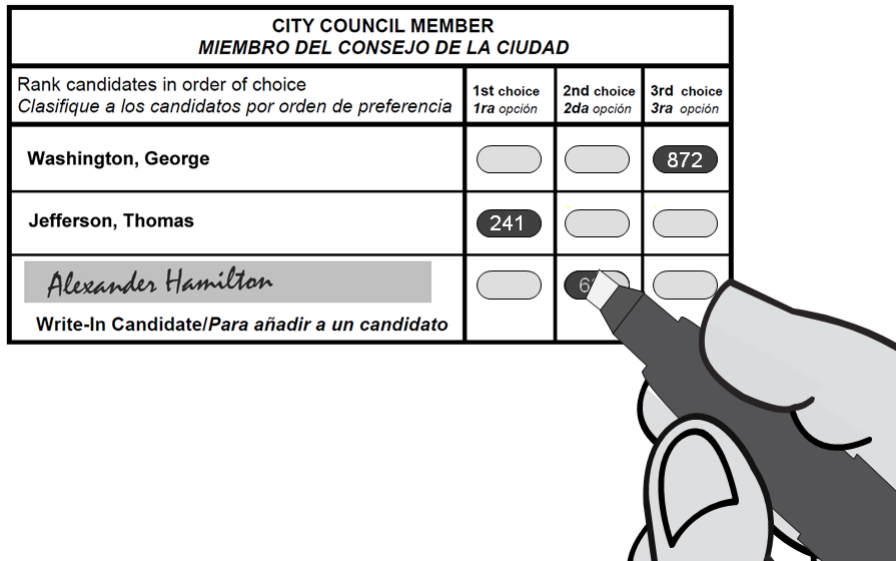
Alignment mark →

2D machine-readable bar code →

1-634527

For voter to look up online → Online Verification Number/
Número de Verificación por Internet

Figure 5.2: An unmarked Takoma Park 2009 ballot for Ward 1 showing instructions in Spanish and English, the options, the circular alignment marks, the 2D barcode, the ballot serial number (on the stub, meant for poll workers to keep track of the number of ballot used) and the online verification number (for voters to check their codes). The true ballot was printed on legal size paper and was hence larger than shown.



(a) Infographic showing the Takoma Park instant-runoff style ballot with Scantegrity II invisible ink confirmation codes.

INSTRUCTIONS FOR VERIFYING YOUR VOTE ON-LINE AFTER YOU RETURN HOME
PARA LAS INSTRUCCIONES EN ESPAÑOL VEA AL DORSO

You have the **OPTION** of verifying your vote on-line after you return home. **It is not necessary to do so.** You may ignore this step entirely; **your cast ballot will be counted whether or not you do this verification.**

If you wish to verify your vote on-line, perform the following steps:

- Fill out your ballot according to the instructions provided on the ballot. "Confirmation numbers" will appear inside the ovals you mark.
- BEFORE YOU CAST YOUR BALLOT** Record the Online Verification Number and the confirmation numbers below, using the narrow tip of the special pen (note that Wards 1-5 will not have a 3rd choice confirmation number for the city council race).

"On-Line Verification Number" from the bottom right corner of your ballot

Confirmation Numbers	1 st Choice	2 nd Choice	3 rd Choice
Mayor	<input style="width: 100%; height: 20px;" type="text"/>	<input style="width: 100%; height: 20px;" type="text"/>	<input style="width: 100%; height: 20px;" type="text"/>
City Council Member	<input style="width: 100%; height: 20px;" type="text"/>	<input style="width: 100%; height: 20px;" type="text"/>	<input style="width: 100%; height: 20px;" type="text"/>

- Cast your ballot as usual using the poll-site scanner. **DO NOT CAST THIS SHEET, but take it home with you.**
- After you have returned home, use a computer with an Internet connection to access the City Clerk's web page: www.takomaparkmd.gov/clerk. Here you will see instructions for verifying that the confirmation numbers you wrote down are correctly recorded. Note that the confirmation numbers are randomly generated and cannot be used to determine your vote.

Thank you for verifying your vote!
The Takoma Park Board of Elections

(b) Verification card for writing down confirmation numbers (true size 8.5" x 11").

Figure 5.3: Supplementary materials provided to the voter.

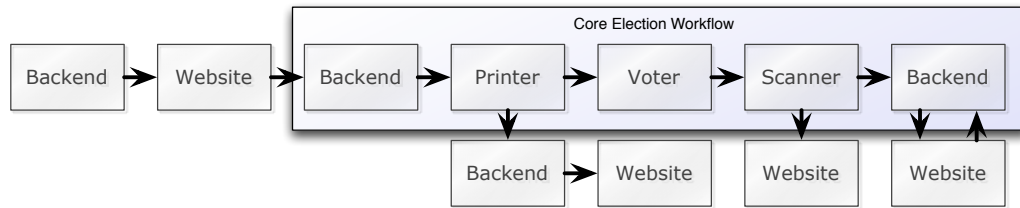


Figure 5.4: **Election Workflow.** The core election work flow in Scantegrity is similar to an optical scan election: a software backend creates ballot images that are printed, used by voters, and scanned. The results are fed to the backend which creates the tally. The audit capacity is provided by 3 extra steps: (1) create the initial digital audit trail and audit a portion of it, (2) audit the ballots to ensure correctness when printing, and (3) audit the final tally.

Digital Audit Trail. Prior to the election, a group of election trustees secret-share a seed to a pseudorandom number generator (PRNG). The trustees then input their shares to a trusted workstation to generate the pseudorandom confirmation codes for all ballots, as well as a set of tables of cryptographic commitments to form the digital audit trail. These tables allow individual voters to verify that their votes have been included in the tally, and allow any interested party to verify that the tally has been computed correctly, without revealing how any individual voter voted.

Auditing. After the election, any interested party can audit the election by using software to check the correctness of the data and final tally on the election website. Additionally, at the polling place on the day of the election, any interested party can choose to audit the printing of the ballots. A print audit consists of marking all of the bubbles on a ballot, and then either making a photocopy of the fully marked ballot or copying down all of the revealed confirmation codes. The ballot ID is recorded by an election judge as audited. After the election, one can check that all of the confirmation codes on the audited ballot, and their correspondence with ballot choices, are posted correctly on the election website.

5.4 Implementation

The election required a cryptographic *backend*, a *scanner*, and a *website*. These 3 components form the basic election system and their interaction is described in Figure 5.4. In addition, Takoma Park required software to resolve write-in candidate selections and produce a formatted tally on election night.

Scantegrity protects against manipulation of election results and maintains, but does not improve, the privacy properties of optical scan voting systems that use serial numbers.

To compromise voter privacy using Scantegrity features, an attacker must associate receipts to voters and determine what confirmation numbers are associated to each candidate. This is similar to violating privacy by other means; for example, an attacker could compromise the scanner and determine the order in which voters used the device, or examine physical records and associate serial numbers to voters. The scanner and backend components protect voter privacy, but the website and the write-in candidate resolver do not because they work with public information only. Each component is written in Java. We describe the implementation and functions of each one in the following sections.

5.4.1 Cryptographic Backend

Implementation. The cryptographic backend that provides the cryptographic verifiable audit data utilizes the Punchscan code base [PH06, ECCP07b] written in Java 1.5 using the BouncyCastle cryptography library.⁶ Key management in the Punchscan backend is handled by a simple threshold [Sha79] cryptosystem that asks for a username and password from the election officials. The code base was adapted to implement the Scantegrity Switchboard functionality (cf. Chapters 3 and 4). Although the resulting software and audit data sets were larger and more complicated to verify than strictly necessary, reusing Punchscan code saved considerable work and time.

Popoveniuc and Vora provide additional detail [PV08], which we will briefly summarize. The Punchscan backend uses a two-stage mix process based on cryptographic commitments published before the election. Each mix, the *left mix* and the *right mix*, takes marked positions as input, shuffles the ballots, and reorders each marked position on each ballot according to a prescribed (pre-committed) permutation. The result is the set of cleartext votes, where position 0 corresponds to candidate 0, 1 to 1, etc. Between the two mixes, for example, position 0 may in fact correspond to candidate 5, depending on the permutation in the *right mix*.

The Punchscan backend partitions [PS07] each contest such that each contest is treated as an independent election with a separate set of commitments. In the case of Takoma Park, each ward race and the mayor's race are treated as separate elections. (The announcement of separate mayoral race vote counts for each ward is required by Takoma Park). The scanner is responsible for creating the input files for each individual election.

Meetings of Trustees. Election officials hold a series of meetings using the backend to conduct an election. Before the election, during *Meeting 1* (Initialization), they choose passwords that are used to generate master key, which in turn is used as a seed for generating all other data (pseudo) random for the election. After each meeting, secret data (such

⁶<http://www.bouncycastle.org>

as the mapping from confirmation codes to candidates) is regenerated from the passwords when it is needed again. In *Meeting 1* the backend software creates a digital audit trail by committing to the Punchscan representation of candidate choices and to the *mixset*: the left and right mix operations for each ballot. Later, during *Meeting 2* (Pre-Election Audit), the backend software responds to an audit of the trail demonstrating that the mixset decrypts ballots correctly. At this time, the backend also commits to the Scantegrity front-end, consisting of the linkage between the Scantegrity front-end and its Punchscan backend used for decryption.

After the election, election officials run *Meeting 3* (Results), publishing the election results and the voted confirmation numbers. For the purposes of the tally audit, the system also publishes the outputs of the left and right mixes. In *Meeting 4* (Post-Election Audit), officials respond to the challenges of the tally computation audit. Either the entire *left mix* or the entire *right mix* operations are revealed, and the auditor checks them against data published in *Meeting 3*.

The *Meeting 4* audit catches, with probability one half, a voting system that cheats in the tally computation. To provide higher confidence in the results, the backend creates multiple sets of left and right mixes; in Takoma Park, we created 40 sets for each election, 20 of which were audited. Given 2 contests per ballot and 40 sets of left and right mixes, there are a total of 160 commitments per ballot in the audit trail, in addition to a commitment per contestant per ballot for each confirmation number (15–18, depending on the Ward).

The implementation uses two classes of “random” number sources. The first is used to generate the digital audit trail, and the second is used for auditing the trail. Both types of sources must be unpredictable to an adversary, and we describe each in turn.

Cryptographic Audit Dataset. The Punchscan backend generates the mixes and commitments using entropy provided by each election official during initialization of the threshold encryption. This provided a “seed” for a pseudorandom number generator (based on the SHA256 hash function).

We also used this random source to generate the confirmation numbers when changing the Punchscan backend to support Scantegrity. Unfortunately, we introduced an error in the generation when switching from alphanumeric to numeric confirmation numbers as a result of findings in the Mock election [SCC⁺10]. This resulted in approximately 8.5 bits of entropy as opposed to the expected 10 bits. We discovered this error after we started printing and it was too late to regenerate the audit trail.

The error increased the chance that an adversary could guess an unseen confirmation code to approximately one in 360 rather than the intended one in 1000: a small decrease in the protection afforded against malicious voters trying to guess unseen codes in order to discredit the system.

Verification. Random numbers are needed to generate challenges for the various auditing steps (print audit, randomized partial checking). These numbers should be unpredictable in advance to an adversary. They should also be “verifiable” after the fact as having come from a “truly random” source that is not manipulable by an adversary.

We chose to use the closing prices of the stocks in the Dow Jones Industrial Average as our verifiable but unpredictable source to seed the pseudorandom number generator (the use of stock prices for this purpose was first described in [CEA07b]). These prices are sufficiently unpredictable for our purposes, yet verifiable after the fact. However, it turns out that post-closing “adjustments” can sometimes be made to the closing prices, which can make these prices less than ideal for our purposes in terms of verifiability.

Scanner Software. The original intent of Scantegrity was to build on top of an existing optical scan system. There was no pre-existing optical scan system in use at Takoma Park, so we implemented a simple system using EeePC 900 netbooks and Fujitsu 6140 scanners.

The scanning software is written in Java 1.6. It uses a bash shell script to call the SANE scanimage program⁷ and polls a directory on the filesystem to acquire ballot images. Once an image is acquired it uses circular alignment marks to adjust the image, reads the barcode using the ZXing QRCode Library,⁸ and uses a simple threshold algorithm to determine if a mark is made on the ballot.

Individual races on each ballot are identified by ward information in the barcode, which is non-sequential and randomly generated. The ballot id in the barcode and the web verification numbers on each ballot are different numbers, and the association between each number type is protected by the backend system. Write-in candidate areas, if that candidate is selected by the voter, are stored as clipped raw images with the ballot scan results. Ballot scan results are stored in a random location in a memory mapped file.

The current implementation of the scanning software does not protect data in transit to the backend, which poses a risk for denial of service. Checking of the correctness of the scanner is done through the Scantegrity audit. The data produced by the scanner does not compromise voter privacy, but—assuming an attacker could intercept scanner data—voter privacy could be compromised at the scanner through unique write-in candidates on the ballot, through a compromised scanner, by bugs in the implementation, or by relying on the voter to make readable copies of the barcode to get a ballot id.

Tabulator/Write-In Software. At the request of Takoma Park we created an additional piece of software, the Election Resolution Manager (ERM), that allows election

⁷<http://www.sane-project.org/>

⁸<http://code.google.com/p/zxing/>

judges to manually determine for each write-in vote what candidate the vote should be counted toward. The other responsibility of the ERM is to act as part of the backend. It collates data from each scanner and prepares the input files for the backend.

To resolve write-ins with this software, the user cycles through each image, and either types in the name of the intended candidate or selects the name from a list of previously identified candidates composed of the original candidates and any previously typed candidate names. The user is not shown the whole ballot, so he does not know what the other selections are on that ballot, or what rank the write-in was given. We call this process *resolving* a vote because the original vote is changed from the generic “Write-In” candidate to the candidate that was intended by the voter. The ERM produces a PDF of each image, the candidate selection for that image, and a unique number to identify the selection.

Scantegrity handles write-in candidates just like other optical scan systems by treating the write-in position as a candidate. Therefore, the backend does not know how each write-in position was resolved, and two results records are created: one with write-in resolution provided by the ERM, and one without write-in resolution provided by the backend.

To check the additional record generated by the ERM, an observer reduces the resolved results record and verifies that the set of resolved ballots is the same as the set of unresolved ballots. To audit that the judges chose the correct candidates for each write-in, the observer refers to the PDF generated during write-in resolution. The PDF allows the observer to reference each resolved ballot entry in the resolved results file and verify that the image was properly transcribed.

One caveat of this approach is that if a write-in candidate wins, a malicious authority could modify these images to change results, but could not deny that the write-in position had received a winning number of votes. This situation would require additional procedures to verify the write-ins (e.g. a hand count, and/or careful audit of the transcriptions by each judge).

Website. Beyond communicating the election outcome itself, the role of the election website is to serve as a “bulletin board” (BB) to broadcast the cryptographic audit data set (i.e., cryptographic commitments, responses to audit challenges, etc). In addition, voters can use this website to check their receipts, and file a dispute if the receipt is misreported. We provided an implementation with these features written in Java 1.6. It used the Stripes Framework⁹ and an Apache Derby database backend.¹⁰ In practice, we only used part of this implementation.

Originally, our plan was to have Takoma Park host the website, but officials chose a hybrid approach where they hosted election information and results. That website would

⁹<http://www.stripesframework.org/>

¹⁰<http://db.apache.org/derby/>

link to our server to provide a receipt checking tool and audit data. After the election, officials would provide us with a copy of the public data files to publish. This decision caused a number of changes to our approach.

We decided to only use the receipt checking code from the implementation, and, to make downloading more convenient for auditors, post all election data on our publicly available subversion repository.¹¹ Additionally, both auditors agreed to mirror the data.

A primary security requirement for the Scantegrity BB is to provide authenticated broadcast communication from election officials to the public. We met this requirement with digital signatures. A team member (Carback) created signed copies of each file with gnupg¹² using his public key from May 28, 2009.

Without authenticated communication, it would be impossible to prove if different results were provided to different people. Our specific approach to the website requires observers to verify signatures and check with each other if they receive identical copies of the data (and verify the consistency of the signatures over time). Our auditors, Adida and Zagórski, performed these actions, but we do not know the extent of this communication otherwise. As usual with our approach to Scantegrity, we are enabling **detection** of errors (genuine or malicious).

There are several potential threats to the bulletin board model—we will briefly enumerate some of them. At a high level, threats pertain primarily to misreporting of results, or to voter identification. With regard to results reporting, an adversary may attempt to misreport results by substituting actual election data with false data. In the event that all parties verify signatures of information they receive, and check consistency with the signed files, incorrect confirmation codes on the bulletin board would be detected by voters, and incorrect computation of the tally by anyone checking the tally computation audit. If the voter checking confirmation codes does not check consistency with the rest of the bulletin board (by, for example, downloading the bulletin board data, checking all the signatures and checking that his or her confirmation code is also correctly noted in the entire bulletin board data) he or she may be deceived into believing their ballot was accurately recorded and counted. Similarly, if the various signatures are not cross checked across individuals or observed over time, an adversary may replace the confirmation codes after they have been checked, or send different ones to voters and to auditors. An adversary may also attempt an identification attack, whereby the objective is to link voter identities with receipt data, such as by recording IP addresses of voters who check their receipts.

¹¹<http://scantegrity.org/svn/data/takoma-nov3-2009/>

¹²<http://www.gnupg.org/>

5.5 The Election

In this section, we describe the election as events unfold chronologically over time.

5.5.1 Preparations Prior to Election Day

Preparations for the election include running the first 2 backend meetings, and creating the ballot.

Election Verifiers/Auditors. The Board of Elections requested cryptographers Dr. Ben Adida (Center for Research on Computation and Society, Harvard University) and Dr. Filip Zagórski (Institute of Mathematics and Computer Science, Wrocław University of Technology, Poland) to perform independent audits of the digital data published by Scantegrity in general, and of the tally computation in particular. Dr. Adida¹³ and Dr. Zagórski¹⁴ maintained websites describing the audits and the results of the audits, and Dr. Adida also blogged the audit.¹⁵ Before the election, Dr. Adida pointed out several instances when the Scantegrity information was insufficient; Scantegrity documentation was updated as a result.

The Board of Elections also requested Ms. Lillie Coney (Associate Director, Electronic Privacy Information Center and Public Policy Coordinator for the National Committee for Voting Integrity (NCVI)) to perform print audits on Election Day. Ms. Coney chose ballots at random through the day, exposed the confirmation codes for all options on the ballot, and kept these with her until after the end of the complaint period, when Scantegrity opened commitments to all unvoted and unspoiled ballots (and hence to all ballots she had audited). Ms. Coney then checked that the correspondence between codes and confirmation numbers on her ballots matched those on the website.

Meeting 1: Switchboard Generation and Commitment. Four election officials (the City Clerk, the Chair, Vice Chair and a member of the Board of Elections: Jessie Carpenter, Anne Sergeant, Barrie Hofmann and Jane Johnson, respectively) were established as election trustees in *Meeting 1*, held on October 12, 2009.

It was explained to the trustees that, through their passwords, they would generate the confirmation codes and share the secret used to tally election results. Further, it was explained that, without more than a threshold of passwords, the election could not be

¹³<http://sites.google.com/site/takomapark2009audit/>

¹⁴<http://zagorski.im.pwr.wroc.pl/scantegrity/>

¹⁵<http://benlog.com/articles/category/takoma-park-2009/>

tallied by Scantegrity, and that if a threshold number of passwords was not accessible (if they were forgotten, for example, or trustees were unavailable due to sickness) the only available counts would be manual counts. A threshold of two trustees was determined based on anticipated availability of the officials, and it was explained that two trustees could collude to determine the correspondence between confirmation numbers and codes, and hence that each trustee should keep her password secret.

The trustees generated commitments to the decryption paths for each of 5000 ballots per ward (for six wards). Scantegrity published the commitments on October 13, 2009 at 12:13 am.

Meeting 2: Pre-election Switchboard Audit. In *Meeting 2*, held on October 14, 2009, trustees used Scantegrity-written code to respond to challenges generated using stock market data at closing on October 14. Half of the ballot decryption paths committed to in *Meeting 1* were opened. Additionally, trustees constructed ballots (associations between candidates and confirmation codes) at this meeting, and generated commitments to them. Scantegrity published the stock market data, the challenges, and the responses.

Ballot Design. The ballot used for the 2009 election was based on ballots used for the 2007 election. We made the conscious choice to modify (as little as possible) a design already used successfully in a past election, and not to use the ballot we had designed for the mock election. The main reason for reusing the ballot design was that it would be familiar to voters. The ballot was required to contain instructions in both English and Spanish: marking instructions, instructions for write-ins, instructions for IRV and any Scantegrity-related instructions (see Figure 5.2).

Printing Ballots. We use “invisible” ink to print the marking positions that reveal confirmation codes to voters. We used refillable inkjet cartridges in multiple color positions of an Epson R280 printer to print confirmation codes. The ink is not actually invisible, but looks like a yellow bubble before marking and a dark bubble with light yellow codes after marking.¹⁶

We initially began printing with 6 printers, but they proved unreliable. It was our expectation that using large amounts of commodity hardware would scale, but it did not. We did not anticipate the number of failure modes we experienced and our printing process was delayed by approximately one and a half days.

¹⁶See <http://scantegrity.org/~carback1/ink> for more information on the printing process

Ballot Delivery. Mail-in (absentee) ballots were delivered to the City Clerk on 16 October. Early, in-person voting ballots were delivered on October 27 for early voting on October 28, and all other ballots a couple of days later on October 30.

Absentee ballots were identical to in-person voting ballots except they did not contain online verification numbers and voters were not given any instructions on checking confirmation numbers online. They were returned by mail in double envelopes and scanned with the early votes. Confirmation numbers for these ballots were, however, made available online after scanning, so that there was no distinction in published data between absentee and in-person voted ballots.

The board decided to issue ballots without confirmation numbers due to the small number of anticipated absentee votes and the costs associated with mailing ballots with special pens. Mailing the ballots with confirmation codes would allow verification of confirmation codes, but opens up new attacks: the possibility of false charges of election fraud by adversaries who might expose confirmation codes and reprint ballots, or use expensive equipment to attempt to determine the invisible codes. Strong verification for absentee ballots is an ongoing research subject within the Scantegrity team.

Early in-person voters used Scantegrity ballots with all Scantegrity functionality, except that the early votes were scanned in after the polls closed on Election Day, and not by voters themselves. Voters were, however, provided verification cards and could check confirmation codes for these ballots online.

Poll Worker Training. Several training sessions were held in the weeks prior to the election. Manuals from the previous election were updated and a companion guide was created with Scantegrity-specific instructions. Election judges were given these two manuals, and a member from our team demonstrated the voting process at one session.

Voter Education. Voter education for this election focused on online verification. Articles in the City newspaper before the real election indicated that voters could check confirmation numbers online; this was also announced on the city’s election website.¹⁷

Scanner Setup. We attempted to minimize, not prevent,¹⁸ the potential for using the wrong software by installing our software on top of Ubuntu Linux on SD flash cards, setting the “read-only” switch on each card, and setting up the software to read and write to USB sticks. We fingerprinted the first card after testing with the sha1sum utility and cloned it

¹⁷<http://www.takomaparkmd.gov/clerk/election/2009/>

¹⁸Scantegrity would detect manipulation at the scanner. A better solution would use trusted hardware technology (e.g. a TPM [FSC09]).

to a second card for the other netbook. Each netbook was set to boot from the card and BIOS configuration was locked with a password.

Both flash cards were checked with the sha1sum utility then placed into the netbook which was placed into a lock box and delivered to Takoma Park. The USB sticks were initialized with scanner configuration files. We uniquely identified each scanner by changing the ScannerID field in the configuration files, then we placed the corresponding USB sticks (3 for each netbook) into the lock box.

Upon delivery of the scanners the day before the election, we gave election officials the lock box keys and showed them how to open the lock boxes. We confirmed with election officials the contents of each box and the officials verified, with our assistance, that the USB memory sticks did not contain any ballot data by looking at the configuration file and making sure the ballot data file was blank.¹⁹ To protect against virus infection on the sticks we set them to read-only for this procedure.

5.5.2 Election Day

On Election Day, November 3, 2009, polls were open from 7 am to 8 pm at a single polling location, the Takoma Park Community Center. Several members of the SVST were present through most of the day in the building in case of technical difficulty. One SVST member was permitted in the polling room at most times as an observer, and a couple of SVST members were present in the vestibule giving out and collecting survey forms through most of the day. Lillie Coney of the Electronic Privacy Information Center, who performed a print audit on the request of the Board of Elections, was present in the polling room through a large part of the day.

Starting the Election. The scanner was the only SVST equipment to set up and it was a turn key system. Election judges needed to plug in the USB sticks and power on the netbooks. The scanner was attached to a scanning apparatus, and cables were run into the lockbox that contained the netbook. When ready, the scanner would beep 3 times. After reading a ballot, the scanner would beep 1 time. During shutdown, the scanner would beep another 3 times. If there were any failure modes the scanner would beep continuously or not beep at all.

Election judges set up the check-in tables, pollbooks, and voting booths. The election started on time.

¹⁹These were the only 2 files on the disk at this time. Additionally, election officials did not check fingerprints on the flash cards. Since no third party had reviewed the code or fingerprinted it they relied on our chain of custody.

Voting. The election proceeded quite smoothly, with very few (small) glitches. An SVST member was able to assist polling officials in fixing a problem with their poll books (not provided by Scantegrity). Voters had some initial problems with the use of the scanner and the privacy sleeve, some seeking assistance from election judges who also had difficulty. After an explanation to the election judges by the Chair of the Board of Elections, the use of the scanner was considerably smoother. With a few ballots, the privacy sleeve was not letting go of the ballots; one ballot was mangled considerably but scanned fine. Seventeen scanned ballots had lines on them that caused the scanner to be unable to read votes, and one ballot had alignment marks manipulated such that it was also unreadable. Images of all unreadable scans are saved, so we were able to manually enter in these votes. Of the seventeen ballots, many ballots had a line in the same location, which is consistent with there being a foreign substance on a ballot put into the scanner. These problems did not affect our ability to count the votes.

During the day, Ms. Coney chose about fifty ballots at random, uniformly distributed across wards, and exposed the confirmation codes for all options for the ballots. A copy of each ballot was made for her to take with her; the copies were signed by the Chair of the BoE.

Towards the end of the day, after the local NPR station carried clips from an interview with the Chair of the Board of Elections and a voter, the polling station saw a large increase in the number of voters, with the line taking up much of the floor outside the polling room. The SVST generated commitments to more ballots than was required. The number of printed ballots ended up being almost twice the number of voted ballots.

Absentee and early voted ballots were scanned in after the closing of polls. Afterward, the scanners were shut down. The chief judge opened each lock box, set all sticks to read only, removed 2 USB sticks (leaving the third with the scanning netbook), and locked the lock box. Our team was given 1 stick for the ERM system. The other was kept by the city.

Meeting 3: Scantegrity-derived Results. Trustees used Scantegrity code to generate results without provisional ballots at about 10 pm. The Chair of the Board of Election announced the results to those present at the polling place at the time (including candidates, their representatives, voters, etc.); this was also televised live by the local TV station. Confirmation codes and the election day tally were posted on the Scantegrity website.

5.5.3 After the Election

On the next day, around 2 pm, results including verified provisional ballots were published. Takoma Park representatives had announced a tally without provisional ballots the night before, and followed with the tally that included verified provisionals in accordance

Mayor	Votes	Ward	Councilor	Votes	Ward	Councilor	Votes
Roger B. Schlegel	664	Ward 1	Josh Wright	434	Ward 4	Terry Seamens	196
Bruce Williams	1000		Write-ins	13		Eric Mendoza	12
Write-ins	17	Ward 2	Colleen Clay	236		Write-ins	2
			Write-ins	15	Ward 5	Reuben Snipper	71
		Ward 3	Dan Robinson	397		Write-ins	10
			Write-ins	34	Ward 6	Navid Nasr	61
						Fred Schultz	138
						Write-ins	0

Table 5.1: The 2009 Takoma Park Election: City certified election results for the Mayoral and City Councilors' races.

with standard Takoma Park procedures. The final *Meeting 3* results were published on November 4th just before midnight.

The number of registered voters were 10,934 and 1728 votes were cast (15.8%). The city-certified final tally for each contest is provided in Table 5.1. In each race, a majority was won after tallying voters' first choices.

Hand Count and Certification. Following a hand count performed by representatives from both the SVST and Takoma Park, the Chair of the Board of Elections certified the results of the hand count to the City Council at 7 pm on November 5. The hand count and the Scantegrity count differed because officials were able to better determine voter intent during the hand count. For example, in the mayoral race, the scanner count determined that 646 votes were cast for candidate Schlegel, 972 for Williams, 15 for various write-in candidates, and 90 were not cast. The certified hand count totals were 664 votes for Schlegel, 1000 for Williams and 17 for write-in candidates. Thus 48 of a total of 1681 votes in this race would not have been counted by a scanner count alone. The discrepancy was caused by voters marking ballots outside of the designated marking areas. Such marks, while not read by the scanner by definition, are considered valid votes by Takoma Park law. Similarly, 8 of a total of 447 votes for Ward 1 council member, 8 of 251 for Ward 2, 16 of 431 for Ward 3, 10 of 210 for Ward 4, 2 of 81 for Ward 5 and 11 of 199 for Ward 6 were added to scanner vote totals after hand counting.

Meeting 4: Post-Election Audit. During *Meeting 4*, held on November 6 at 6 p.m., trustees used Scantegrity-written code to reveal all codes on voted ballots, and to reveal everything for all the ballots that were not spoiled or voted upon. Trustees also responded to pseudo-random challenges generated by stock market results at closing on November 6. Scantegrity published all data on November 7th around 9am. While the SVST could have chosen to use closing data on an earlier date, such as November 4 or November 5,

which could have been more stable, the team chose to stick to its earlier-announced plan (of using the freshest stock market data) for the sake of consistency.

On November 9, 2009, Dr. Adida and Dr. Zagórski independently confirmed that Scantegrity correctly responded to all digital challenges. In particular, they confirmed that the tally computation audit data was correct. Both made available independently written code on their websites that voters and others could use to check the tally computation commitments.

Confirmation Code Dispute Resolution. The period for complaints regarding the election (including complaints about missing confirmation codes) expired at 6 pm on November 6. The Scantegrity website recorded 81 unique ballot ID verifications, of which about 66 (almost 4% of the total votes) were performed before the deadline. The SVST was also told by a BoE member that at least a few voters checked codes on auditor websites. Both Dr. Adida and Dr. Zagórski made the confirmation codes available on their websites after the election.

The number of voters who checked their ballots on-line before the Takoma Park complaint deadline (66), while not large, was sufficient to have detected (with high probability) any errors or fraud large enough to have changed the election outcome. (Detailed calculations omitted here; these calculations are not so simple, due to the use of IRV.)

Scantegrity received a single complaint by a voter who had trouble deciphering a digit in the code and noted it as “0,” while the Scantegrity website presented it as “8.” The voter requested that codes be printed more clearly in the future. He also stated that if he were not a trusting individual, he would believe that he had proof that his vote was altered.

All codes for all voted ballots were revealed after the dispute resolution period, and all commitments verified by two independent auditors, Dr. Adida and Dr. Zagórski. Hence, the probability that the code was in error is very small, albeit non-zero. Scantegrity does not believe the code was in error, and there were no other complaints regarding confirmation numbers.

Audit of Ballot Printing. Dr. Zagórski provided an interface allowing Ms. Coney to check the commitments opened by Scantegrity in Meeting 4 against the candidate/confirmation code correspondence on the ballots she audited. In her report [Con09], she confirmed that the correspondence between confirmation numbers and candidates on all the printed ballots audited by her was correctly provided by the interface.

Followup. The Board of Elections and an SVST representative met to discuss the election and opportunities for improvement several weeks after the election. Both sides were

largely satisfied with the election. Indeed Scantegrity was successfully used in the following municipal election of Takoma Park in November 2011.

5.6 Surveys and Observations of Voter Experiences

To understand the experiences of voters and poll workers, we timed some of the voters as they voted, asked voters and poll workers to fill out two questionnaires, and informally solicited comments from voters as they left the precinct building. Approved by the Board of Elections and UMBC's Institutional Review Board, our procedures respected the constraint of not interfering with the election process. This section summarizes the results of our observations and surveys.

Timing Data. Sitting unobtrusively as official observers in a designated area of the polling room for part of the day, two helpers (not members of the Scantegrity team) timed 93 voters as they carried out the voting process. Using stopwatches, they measured the number of seconds that transpired from the time the voter received a ballot to the time the voter began walking away from the scanner.

Voting times ranged from 55 secs. to 10 mins. (the second-longest time was 385 secs.), with a mean of 167 secs. and a median of 150 secs. On average, voters who appeared older took longer than voters who appeared younger. Most of the time was spent marking the ballot. The average time to vote was significantly faster than during the April 2009 mock election, when voters took approximately 8 mins. on average due primarily to scanning delays [SCC⁺10].

The observers noted that many voters did not fully use the privacy sleeve as intended, removing the ballot before scanning rather than inserting the privacy sleeve with the ballot into the scanning slot. Two of the 93 observed voters initially inserted the privacy sleeve upside-down, causing the ballot not to be fed into the scanner (even though the scanner could read the ballot in any orientation). A few ran into difficulties trying to insert the sleeve with one hand while holding something else in the other hand.

Election Day Comments From Voters. As voters left the precinct building, members of the Scantegrity team conducting the written surveys, and a helper (a usability expert who is not a member of the Scantegrity team) solicited comments from voters with questions like, "What did you think of the new voting system?" The helper solicited comments 1:30–3:00pm and 7–8pm. A common response was, "It was easy."

Quite a few voters did not understand that they could verify their votes online and that, to do so, they had to write down the code numbers revealed by their ballot choices. Some

explained that they intentionally did not read any instructions because they “knew how to vote.” Others failed to notice or understand instructions on posters along the waiting line, in the voting booth, on the ballot, and in the Takoma Park Newsletter.

In response, later in the day, we announced to voters as they entered the building that there is a new system; to verify your vote, write down the code numbers. These verbal announcements seemed to have some positive effect, and there were fewer voter comments expressing lack of awareness of the verification option after we began the announcements. Nevertheless, some voters still were unaware of the verification option. It was a humbling experience to see first-hand how difficult it can be to get across the most basic points effectively, especially the first time a new system is used.

Some of the voters complained about the double-ended pen, not knowing which end to use, or having trouble writing in candidates with the chisel-point (the narrow point was intended for write-ins). A small number of voters had difficulty seeing the code numbers, perhaps largely because repeatedly pressing too hard could erode the paper. A few voters expressed concern about the difficulty of writing down the code numbers, had the ballot been much longer or had there been a large number of competing candidates.

Many voters expressed a strong confidence in the integrity of elections, while a small minority expressed sharp distrust in previous electronic election technology. These feelings seemed to be based more on a general subjective belief rather than on detailed knowledge of election procedures and technology. Similarly, those expressing strong confidence in Scantegrity seemed to like the concept of verification but did not understand in detail why Scantegrity provides high outcome assurance.

Survey of Voter Experiences. As voters were leaving the precinct, we invited them to fill out two one-sided survey forms: a field-study questionnaire, and a demographics questionnaire. The field study asked voters about the voting system they just used, with most answers expressed on a seven-point Likert scale. The last question invited voters to make any additional suggestions or comments. Each pair of forms had matching serial numbers to permit correlation of the field study responses with demographics. 271 voters filled out the forms.

Fifty-one voters wrote comments on the questionnaires, often pointing out confusion about various aspects of the process but with no consistent theme. (1) Some were unaware of verification option. (2) Some did not realize they were supposed to write down code numbers. (3) Some found the pens confusing to use: they did not realize that the pens would expose code numbers, and they did not know which end to use. (4) Some found code numbers were hard to read. (5) Some did not understand how to mark an IRV ballot. (6) Some did not know how to place the ballot into the scanner. (7) One had no difficulty but wondered if seniors or people who speak neither English nor Spanish might

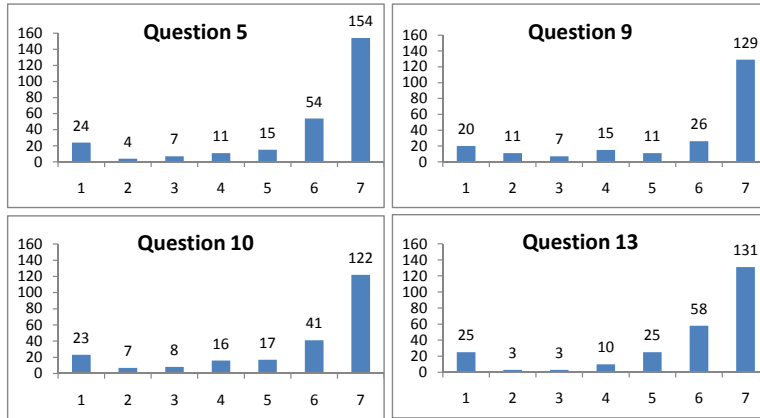


Figure 5.5: Voter responses to Survey Questions 5, 9, 10, 13 from all 271 voters completing the survey. Using a seven-point Likert scale, voters indicated how strongly they agreed or disagreed with each statement about the voting system they had just used (1 = strongly disagree, 7 = strongly agree). Each histogram shows the number of voters responding for each of the seven agreement levels. The four questions shown are the following: (5) Overall, the voting system was easy to use. (9) I have confidence that my receipt by itself does not reveal how I voted. (10) The option to verify my vote online afterwards increases my confidence in the election results. (13) I have confidence in this voting system.

have difficulties. (8) One wondered if the government might be able to discern his vote by linking his IP address used during verification with his ballot serial number and noting the time that he was issued a ballot (this may be possible if the cryptography is broken or in other scenarios, but it would be more direct to have the scanner log how he voted). (9) Many suggested that it would have been helpful to have better instructions, including instruction while they wait in line.

Figure 5.5 shows how voters responded to four questions from the field study questionnaire. These results strongly show that voters found the voting system easy to use (Question 5), and that they had confidence in the system (Question 13). Question 10 showed that the option to check votes on line increased voter confidence in the election results. Question 9 showed that voters had confidence that the receipt alone did not reveal how they voted; this finding is notable given that it is widely suspected that many people erroneously believe that all E2E receipts reveal ballot choices.

Survey of Poll Worker Experiences. Each of the twelve poll workers was given an addressed and stamped envelope with two questionnaires (field study and demographics) to fill out and mail to the researchers after the election. The field study focused on their experiences administering Scantegrity, with most answers expressed on a seven-point Likert scale. This questionnaire also included four open-ended questions. Each pair of forms had matching serial numbers. Five forms were returned.

Poll workers noted the following difficulties. (1) There was too much information. (2) Some voters did not understand what to do, including how to create a receipt. (3) Some voters did not understand how to mark an IRV ballot. (4) The privacy sleeve was hard to use with one hand. (5) The double-ended pens created confusion. (6) Voters, poll workers, and the Scantegrity team have different needs. One wondered if Scantegrity was worth the extra trouble.

They offered the following suggestions: (1) Simplify the ballot. (2) Provide receipts so that voters do not have to copy code numbers. (3) Develop better pre-election voter education.

5.7 Discussion and Lessons Learned

Overall, this project should be deemed a success: the goals of the election were met, and there were no major snafus. Many aspects of the Scantegrity design and implementation worked well, while some could be improved in future elections.

Technological Challenges. Perhaps the most challenging aspect for future elections is scaling up ballot printing. The printers we used were not very reliable.

Variations on the Scantegrity design worth exploring include the printing of voter receipts (rather than having voters copy confirmation codes by hand)—there are clearly security aspects to handle if one does this. The design should also be extended for better accessibility. The special pen might be improved by having only a single medium-tip point, rather than two tips of different sizes. The scanning operation and its interaction with the privacy sleeve should be studied and improved.

The website, while sufficient, might utilize existing research in distributed systems to reduce the requirements on observers and voters to continually monitor the site for updates or modifications. The scanner could also be improved with more sophisticated image analysis, and also to better handle unreadable ballots. It only occurred to us after the election that the write-in resolution process could have greater utility if it were expanded to deal with unreadable and unclear ballots.

Real World Deployment of Research Systems. As is common with many projects, too much was left until the last minute. Better project management would have been helpful, and key aspects should have been finalized earlier. Materials and procedures should be more extensively tested beforehand.

One of the most important lessons learned is the value of close collaboration and clear communication between election officials and the election system providers (whether they be researchers or vendors).

Another lesson learned is that it is both important to provide voters with clear explanations of the new features of a voting system, and to do so efficiently, with minimal impact on throughput. Resolving the tension between these requirements definitely needs further exploration. For example, it might be worthwhile to have an instructional video explaining the Scantegrity system that voters could watch as they come in. The permanent adoption of Scantegrity II in a jurisdiction would, however, alleviate the educational burden over time, as voters learn the system’s features in successive elections.

Comparison With Post-election Audits. It is interesting to compare Scantegrity with the other major technique for election outcome verification: post-election audits. Because these audits do not allow anyone to check that a particular ballot was counted correctly, they do not provide the level of integrity guarantee provided by Scantegrity.

Post-election audits, even those with redundant digital and physical records like optical scan systems, only address errors or malfeasance in the counting of votes and not in the chain of custody.²⁰ In contrast, end-to-end voting systems such as Scantegrity provide a “verifiable chain of custody.” Voters can check that their ballots are included in the tally, and anyone—not just a privileged group of auditors—can check that those ballots are tallied as intended.

It must be admitted, however, that the additional integrity benefits provided by Scantegrity II come at the cost of somewhat increased complexity and at the cost of an increased (but manageable) risk to voter privacy (since ballots are uniquely identifiable). That said, some jurisdictions and/or election systems require or use serial numbers on ballots anyway, and we have proposed several possible approaches to appropriately destroy or obfuscate serial number information. Furthermore, as a corollary to the work of Calandrino et al. [CCF11], a voter wishing to “fingerprint” a ballot can do so without being detected in current paper ballot systems simply by marking ovals in distinctive ways.

5.8 Concluding Remarks

The successful use of the Scantegrity II voting system in the Takoma Park election of November 3, 2009 demonstrates that voters and election officials can use sophisticated

²⁰Having multiple records may make an attacker’s job harder, but note that the attacker only has to change the record that will ultimately be used and/or trusted (not necessarily both). Also, redundancy can work against a system, as changing a digital record in an obviously malicious way may allow time for a more subtle manipulation of the physical record.

cryptographic techniques to organize a transparent secret ballot election with a familiar voting experience. The election results show considerable satisfaction by both voters and pollworkers, indicating that end-to-end voting technology has matured to the point of being ready and usable for real binding governmental elections. Finally, as a testament to the success of the 2009 election, Takoma Park and the Scantegrity research team continued their partnership, where Scantegrity was deployed for a second time in that city for their municipal election of 2011.

The focus of Part I has been squarely on providing verifiability (a secondary task) without interfering with voting/election administration (the primary task). But there is a fundamental difference between making something *verifiable*, and having it be *verified*. With the advances of Part I in hand, the focus of Part II will be to foster broader and deeper participation in election verification.

Part II

Technical Simplifications for Election Auditors

Chapter 6

Aperio

For a voting system to be viable, reasonable ordinary people have to be able to understand and authenticate it.

Bev Harris [Row08]

This chapter is adapted from published work co-authored with Jeremy Clark, and supervised by Carlisle Adams [ECA08].

6.1 Introductory Remarks

The work of Scantegrity II at Takoma Park demonstrated that cryptographic election verification could be refined to the point of being feasible in a governmental election. A major criticism still faced by Scantegrity, however, is that cryptographic election verification is fundamentally too conceptually complex for the average voter. It became an important research objective, therefore, to design an end-to-end system that could abstract out the cryptographic primitives in favor of a more pedagogically friendly (less technical) analog.

As a starting point consider three proposals made by Rivest and Smith [RS07] for such systems that do not directly utilize cryptography. The first two proposals (ThreeBallot and VAV) have strongly counter-intuitive ballot marking procedures,¹ and relies on trusted hardware to validate ballots. The third and more elegant proposal, Twin, requires no

¹ThreeBallot, for example, requires a voter to cast some “votes” for candidates *other* than their preference, while VAV introduces the notion of an ‘anti-vote.’

electronic equipment or special ballot marking or tabulation procedures, although does carry an inherent custody assumption that the “floating receipt” being issued to the voter is a *valid copy* of the ballot of another (anonymous) voter.

In this chapter we describe Aperio, a simple paper-based election integrity verification mechanism with a straightforward paper ballot, and with end-to-end integrity properties that do not rely on chain-of-custody assumptions. Instead we follow an approach similar to Moran and Naor [MN05] based on tamper-evident envelopes, which has the advantage of being a strong physical analog to cryptographic commitments, as well as being an item commonly used in conventional elections.

Contributions. The contributions of this chapter include the Aperio election verification protocol, that offers:

- A conventional hand-counted paper ballot,
- An end-to-end verifiable audit trail based on tamper resistant envelopes rather than cryptographic assumptions,
- A close physical analog to commitment-based cryptographic schemes.

6.2 Basic Paper Scheme

6.2.1 Ballot Format

In a conventional paper-ballot election, a voter is issued a single sheet of paper—a ballot. Under the Aperio scheme, a voter is instead issued a “ballot assembly,” which consists of a paper ballot sheet, a receipt sheet and any number of audit sheets stacked and joined in such a way that only the top ballot layer is visible to the voter. These layers are defined as follows:

- **A ballot:** used to describe any paper *Australian ballot*² with the specific property that the list of candidates or proposals is printed in an independently random order across the set of ballots in an election,

²“An official ballot printed at public expense on which the names of all the candidates and proposals appear and which is distributed only at the polling place and marked in secret.” Merriam-Webster Dictionary.

- **A receipt:** used to describe a sheet of paper, equivalent in size and layout to a “ballot,” but without a candidate list, and additionally a unique serial number,
- **An audit sheet:** is used to describe a sheet of paper, equivalent in size and layout to a “ballot” but without a candidate list. It does not contain a serial number, but has pre-printed regions in which to write one. Additionally provided is a region in which to mark a “commitment reference number.”

For simplicity of the following description, we will consider the base-case ballot assembly, which includes *two* audit sheets. We will refer to the assembly’s layers by a standard office-paper color pallet, in which the ballot, receipt, and two audit sheets are assigned the colors “white,” “canary,” “goldenrod,” and “pink,” respectively (see Figure 6.1). The sheets are stacked in such a way that voter-made marks on the ballot sheet will be transferred to the other sheets using carbon-copy, or alternatively NCR brand (carbonless copy) paper (see Figure 6.2).

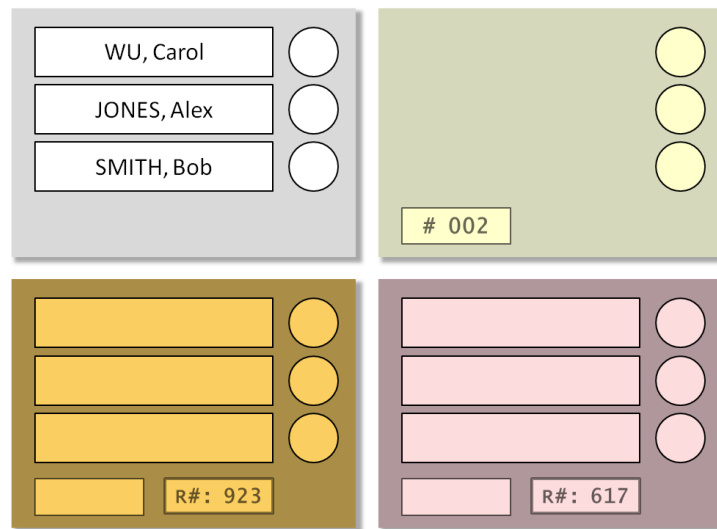


Figure 6.1: **Ballot Assembly:** (**Top left**) Paper “Australian” ballot with random candidate order (*white*). (**Top right**) Receipt with unique serial number (*canary*). (**Bottom left**) Audit sheet with “commitment reference number” (*goldenrod*). (**Bottom right**) Audit sheet with independently assigned “commitment reference number” (*pink*).

6.2.2 Initial Setup

As in other E2E systems, the entity responsible for printing ballots is generally entrusted with voter privacy. For simplicity we describe the following operations as being performed

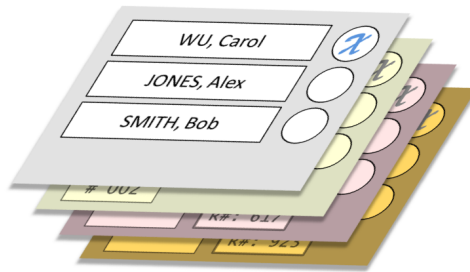


Figure 6.2: **Ballot Assembly (Exploded View)**: Marks made on the top “ballot” layer are transferred to the “receipt” and “audit sheet” layers using “carbon-copy” style paper.

by a single privacy-entrusted entity, although as we will see in Part III, protocols for printing ballots in threshold-trust/oblivious manner, are available.

Generating Ballots. In order to preserve voter privacy, the association of candidate order and serial number must be secret and arbitrary, such that knowledge of one in no way implies knowledge of the other. Consider a stack of b ballots, each with an independently randomly printed candidate order. Likewise consider a stack of b receipts each with unique serial number. An arbitrary association can be formed by drawing the top ballot and receipt sheets from the respective stacks and joining (*e.g.*, stapling) them together. Additionally the (blank) audit sheets are joined to the ballot and receipt sheets, constituting a specific instance of a ballot assembly. This is repeated to create b independent ballot assemblies.

Generating Commitment Lists. A “commitment list” is defined as a list of b rows pre-printed with a monotonically increasing set of b “commitment reference numbers.” Next to each commitment reference number is a region, initially blank, that will contain an associated value. We define two types of commitment lists:

- **Receipt commitment lists** contain a set of b distinct commitment reference numbers, each with a (randomly) associated *serial number*,
- **Ballot commitment lists** contain the same set of b distinct commitment reference numbers, each with a (randomly) associated *candidate list ordering*.

To generate the commitment lists, we begin by considering a particular audit trail color (*e.g.*, pink). The *pink receipt commitment list* and *pink ballot commitment list* are generated as follows:

1. A ballot assembly is drawn from the stack and the serial number s is noted. A non-replaced random number $i \in [b]$ is selected (*e.g.*, on a slip of paper drawn from a hat),
2. The pink audit sheet of the ballot is exposed, and the number i is written in the commitment reference number space,
3. On the *pink receipt commitment list*, the number s is written in the blank space beside commitment reference number i ,
4. On the *pink ballot commitment list*, the candidate order o is written in the blank space beside commitment reference number i .

These steps are performed on all ballot assemblies. The pink audit trail is now complete and the *goldenrod receipt commitment list* and *goldenrod ballot commitment list* (and any additional audit trail colors) can be generated in the same manner. An example ballot assembly and corresponding entries in the commitment lists are depicted in Figure 6.3.

Receipt Commitments		Ballot Commitments	
Ref#	Serial #	Ref#	Candidate List
...
923	002	923	WU, JONES, SMITH
...

Receipt Commitments		Ballot Commitments	
Ref#	Serial #	Ref#	Candidate List
...
617	002	617	WU, JONES, SMITH
...

Figure 6.3: **Commitment Lists:** For *each* audit trail (goldenrod and pink in this case) two commitment lists are generated – a **receipt** list and a **ballot** list, each randomly associating serial numbers and respectively candidate orderings with a distinct commitment reference number.

Committing. For an election with two audit trails (pink and goldenrod), the election trustees generate the *pink receipt*, *pink ballot*, *goldenrod receipt* and *goldenrod ballot commitment* lists. They lock these values in time (*i.e.*, commit to them) through the following procedure:

1. Each commitment list is placed in its own appropriately labeled tamper-evident document envelope and sealed,
2. The trustees present the sealed envelopes to the verifiers, who are given the opportunity to inspect the exterior of the envelope and sign on the flaps,
3. The envelopes are returned into the custody of the trustees.

6.2.3 Print Audit Selections

In order to ensure the commitment reference numbers of the ballot assemblies point to the same candidate orderings/serial numbers appearing on the ballot and receipt layers, some ballot assemblies will be selected by the verifiers for a “print audit.” The procedure could vary between jurisdictions, but one recommendation would be for the print audit selections to be made in conjunction with the random spot checks of the poll registration book at the polling place during election day (as is the procedure in many paper ballot elections today). A verifier would select a ballot at random from the stack of ballots, and the poll worker would mark the ballot as spoiled (*e.g.*, by punching a hole through the layers). The spoiled ballot would then be given to the verifier to retain for a later auditing procedure.

6.2.4 Voting

Voting is conducted in accordance with the jurisdiction’s procedures for paper ballot elections. An eligible and authenticated voter is issued a ballot assembly by a poll official and is directed to a voting booth. The voter marks the ballot assembly as they normally would in any conventional paper ballot election. They return the ballot assembly to the poll official, who first inspects the assembly to ensure the respective layers are still sealed together. The official then separates and distributes the ballot layers in the following way: the ballot layer is cast in the ballot box. The receipt (canary) layer is issued to the voter as a receipt. The pink and goldenrod audit sheets are cast into “pink” and “goldenrod” audit boxes respectively.

6.2.5 Election Outcome

After the close of the polls, the election results are tallied normally, in accordance with the pre-existing procedures of the jurisdiction for paper ballot elections using contents of the ballot box and is referred to as the “official tally.” At the close of the polls, the “pink” and “goldenrod” audit boxes are relinquished into custody of the verifiers.

6.2.6 Decomitting

A coin is flipped in public. If the outcome is heads, the pink audit box is selected to become the ballot audit trail and the goldenrod audit box is selected to become the receipt audit trail. If the outcome is tails, the opposite envelopes are selected. Trustees respond by releasing (*i.e.*, decommitting) the corresponding commitment envelopes through the following procedure:

1. Trustees relinquish selected commitment envelopes into the custody of the verifiers,
2. Verifiers inspect envelopes for the presence of their signatures on the flap and for the absence of evidence of physical tampering of the envelope,
3. Trustees destroy (*e.g.*, shred) the remaining unselected commitment envelopes.

In the following explanation of the receipt and tally audit, for clarity, we will assume a case in which *heads* was the outcome of the coin flip—meaning goldenrod and pink were selected to become the receipt and ballot audit trails, respectively.

6.2.7 Receipt Audit

Using the contents of the goldenrod audit trail box in conjunction with the goldenrod receipt commitment list, a receipt trail can be reconstituted in the following way (see Figure 6.4):

1. A goldenrod audit sheet is drawn from the goldenrod audit trail box. The commitment reference number $i \in [b]$ is noted,
2. The i -th row of the *goldenrod receipt commitment list* is consulted, and corresponding serial number s is noted,
3. The number s is written into the blank serial number space on the audit sheet.

These steps are performed on all goldenrod audit sheets. The reconstituted receipts can be cross-referenced against voter receipts to ensure the records match. Voters could optionally give their receipts to the verifiers to cross-reference on their behalf, or alternatively the verifiers could publish the reconstituted receipt trail in a public venue (*e.g.*, newspaper) with which voters could check for themselves.

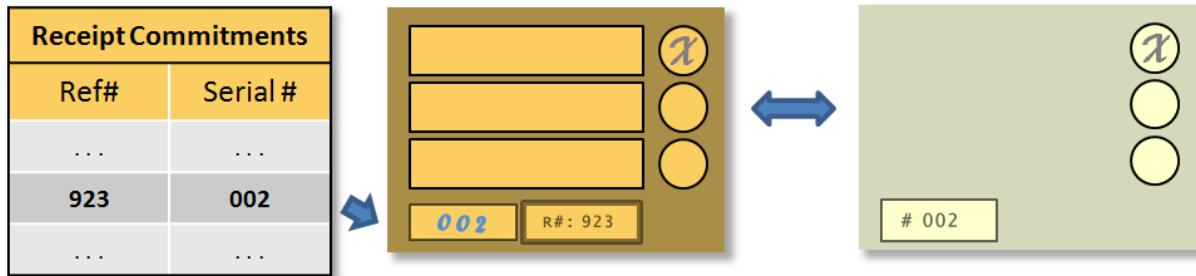


Figure 6.4: **Reconstituting the Receipt Audit Trail:** The decommitted (goldenrod) receipt commitment list (**Left**) can be used to reconstitute receipts from the corresponding (goldenrod) audit trail (**Middle**). The reconstituted receipt audit trail can be cross-referenced against voter receipts (**Right**).

6.2.8 Tally Audit

Using the contents of the pink audit trail box in conjunction with the pink ballot commitment list, a ballot can be reconstituted in the following way (see Figure 6.5):

1. A pink audit sheet is drawn from the pink audit trail box. The commitment reference number $j \in [b]$ is noted,
2. The j -th row of the *pink ballot commitment list* is consulted, and corresponding candidate list ordering o noted,
3. The candidates are written in order o into the blank candidate name spaces on the audit sheet.

These steps are performed on all pink audit sheets. The reconstituted ballots can be tallied and cross-referenced against the official tally to ensure a match.

6.2.9 Print Audit

Recalling the randomly selected (spoiled) ballots from section 6.2.3, the correctness of a ballot assembly's printing can be verified in the following way:

1. For a given ballot assembly, candidate order o and serial number s are noted,
2. The ballot assembly's pink and goldenrod audit layers are reconstituted into ballot and receipt audit layers as described in sections 6.2.7 and 6.2.8 to recover the o' and s' pointed to by the respective commitment reference numbers,
3. The printing of this given ballot assembly is correct if $o = o'$ and $s = s'$.

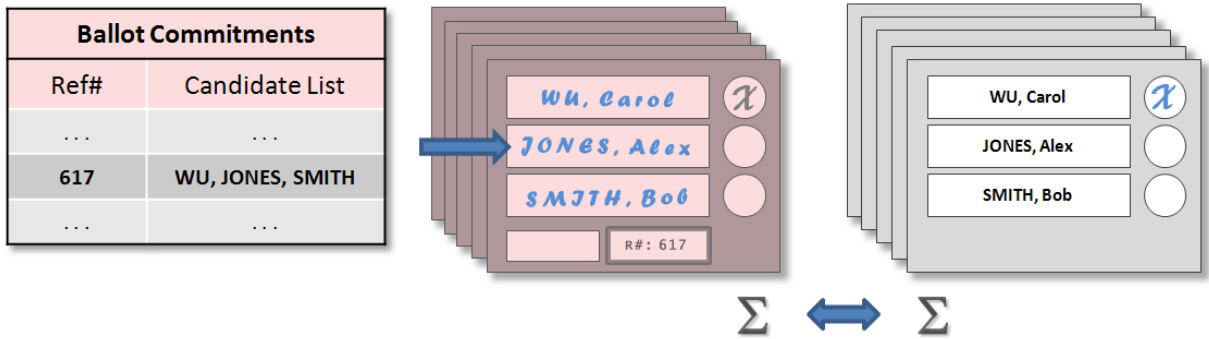


Figure 6.5: **Reconstituting the Ballot Audit Trail:** The decommitted (pink) ballot commitment list (**Left**) can be used to reconstitute ballots from the corresponding (pink) audit trail (**Middle**). The reconstituted ballot audit trail can be tallied and the totals cross-referenced against the official tally (**Right**).

6.3 Security Analysis

In this section, we describe how Aperio meets the E2E integrity criteria as outlined in Chapter 2. Further, we analyze the attack vectors that an adversary could use to attempt to corrupt the results of an election and demonstrate the protections offered by Aperio to thwart these attacks.

6.3.1 A Positive Assertion of Security

Let an unmarked Aperio ballot assembly be the tuple $\langle o, s, c_p, c_g \rangle$ for candidate order, serial number, commitment reference number of the pink sheet, and commitment reference number of the goldenrod sheet. Let ρ denote the position marked by the voter on each element of the ballot assembly. For the following discussion, again consider the instance in which the *pink receipt commitment list* and the *goldenrod ballot commitment list* were selected to be decommitted (although the following security properties are invariant to any particular selection). The audit process establishes the following facts:

1. The voter's receipt contains $\langle s, \rho \rangle$. By matching the voter's receipt to the *receipt commitment list*, it can be verified that ρ' (of row s of the receipt commitment list) matches the ρ on the voter's receipt. Therefore, the voter's mark is included unmodified in the collection of ballots—the first E2E criterion,
2. The print audit verifies that s and c_p printed on a ballot are the same as in the commitment reference sheet and additionally,

3. Verifies o and c_g are the same as on the ballot reference sheet,
4. Since 2 and 3 are dependent on a random decision, it is probable that s and c_g also are consistent between the printed ballots and reference sheets, and additionally,
5. It is probable that o and c_p also are consistent between the printed ballots and reference sheets. If the printed ballots are not consistent, this would be detected with probability $1 - (1 - Y)^{x-1}$ where Y is the percentage of receipts checked and x is the number of audit sheets,
6. By combining facts 2 and 5, or similarly 3 and 4, we infer that s and o on the sheets are consistent with what the voter saw in the polling booth,
7. By combining 1 and 6, the voter is assured that the same ρ at s on their receipt is in the ballot commitment list somewhere beside the same o that was on their ballot,
8. Finally, given 7, the voter can generate a correct tally for all votes using the ballot reference sheet proving that the collection of ballots is tallied correctly—the second property of an E2E election.

The indirectness of this proof prevents the voter from proving which candidate they voted for to a coercer or someone wishing to purchase their vote. The tally that was generated to provide fact 8 can also be compared to the official tally generated using the original paper ballots for additional assurance.

6.3.2 Prevented Attacks

In addition to the positive security properties already outlined, it may be also useful to demonstrate a set of attacks it is not susceptible to.

- **Adding or removing ballots.** In order to increase the number of votes for a candidate, an adversary may “stuff” the ballot box with extra ballots for their candidate of choice. Alternatively, given that elections are local events and that correlations often exist between a locality and a political preference, an adversary may attempt to destroy cast ballots in a discriminatory manner by choosing locations based on expected political preference. We assume that a voter registration list is maintained with the number of voters who cast ballots in the election. This information may also be corroborated by election observers. The number of ballots and audit sheets should be identical between them and should also be identical to the total voters on the registration.

- **Modifying ballots.** A more sophisticated adversary would avoid upsetting the number of cast ballots by either replacing existing ballots with new ballots marked for their candidate or modifying the marks on existing ballots. Since a ballot assembly is distributed between distinct boxes, the tuple $\langle o, s, c_p, c_g \rangle$ becomes unlinked and subsequently unassociable once cast into the respective boxes. If an adversary modifies or replaces o on a ballot, then the audit tally will not match the tally generated from adding up all the top layers. Its not in the interest of an adversary to modify c_p or c_g , as there is no way of knowing which candidate a vote is for and which it will be mapped to if modified. Furthermore, there is only a 50% chance that the tally will change at all—alternatively the box will be used to decommit receipts. In either case, such modifications will be detectable either because the tallies will not match, or the receipts will not match. Since c_g and c_p hide o , the voter cannot change both an o on a ballot and, say, c_g on a goldenrod audit sheet in a way that consistently changes both tallies, should the goldenrod ballot commitment list be opened.
- **Misprinting ballots.** An adversary could also misprint ballot assemblies before voting day, or mix-and-match sheets from one ballot assembly with sheets of another. Most of these attacks will not affect the tally (only the receipts) and there is only one method for potentially modifying the tally in an undetectable way: switch both o and one audit sheet, say c_g , with another ballot with a different order o' , where $o \neq o'$. If the *goldenrod ballot commitment list* is selected to be decommitted, the two tallies will match, otherwise the attack will be detected. Furthermore, since the adversary has no guarantee of which voter will receive the misprinted ballot, votes cannot be predictably directed to a particular favored candidate. This attack is marginal and can be further mitigated by using more than two audit layers, which exponentially decreases the probability of a successful execution of this attack.
- **Forced randomization attack.** An adversary could coerce a voter into returning with a receipt with a particular position marked. Since the adversary has no knowledge of o , this is equivalent to forcing the voter to throw away their vote by voting for a random candidate as demonstrated in [PS08]. Since a random vote will be given to each candidate with equal probability, coercing a random votes will have the same effect as coercing a voter to not vote at all—the latter likely being easier to execute.
- **Chain voting.** An adversary who can capture an unmarked ballot can execute a chain voting attack where they fill out the unmarked ballot for a candidate they choose, then coerce a voter into casting the adversary's ballot in place of the unmarked ballot the voter receives, and demand that the voter return to them the voter's unmarked ballot so they can execute the attack *ad infinitum*. This can be mitigated by the use of a detachable, uniquely marked “counterfoil.” As the voter registration official issues the ballot, the number n on the counterfoil is noted. When

the voter returns, the official verifies the presence of the counterfoil and notes the number n' . If $n = n'$, the voter has not exchanged ballots. The counterfoil is then detached and discarded, and the ballot assembly and the casting procedure continues as in section 6.2.4. As an option, the counterfoil could be additionally utilized to initially seal the ballot layers together to prevent anyone from viewing s , c_g and c_p beforehand.

- **Pattern Attacks.** A voter wishing to sell their vote could mark their ballot in a pattern that is likely to be uniquely identifiable and then point out its location to the buyer in the ballot commitment list. This can be mitigated by partitioning the audit into subaudits: *i.e.*, having separate commitment lists for each contest or small collections of contests. Exactly how to partition a ballot to marginalize a pattern attack can be determined statistically—*i.e.*, [HSS09].

6.4 Privacy

In this section we describe how the scheme protects voter privacy. With the exception of the privacy entrusted entity that generates and prints ballot assemblies, no information about how a voter votes becomes known to the other entities (*i.e.*, voter, poll officials, verifiers) with the following justification:

- **Ballot assembly.** It is easy to see that through the decommitting process, *i.e.*, $\langle c_p, c_g \rangle$ opens to $\langle o, s \rangle$, and therefore $\langle c_p, c_g, \rho \rangle \Rightarrow \langle o, s, \rho \rangle$, which is the basis of the print audit of *spoiled* ballot assemblies. Privacy is preserved on these ballot assemblies given $\rho = \emptyset$ (*i.e.*, is unmarked). Assuming poll procedure is followed, then $\langle c_p, c_g \rangle$ will be physically unlinked by the poll official and logically unlinked by the mixing in the audit trail ballot boxes. It is central to voter privacy that neither the voters, poll officials nor verifiers see $\langle c_p, c_g \rangle$ leading up to, and during, the ballot casting process. The assembly layers might be sealed (*e.g.*, glued) together as previously suggested by a tear-off “counterfoil.”
- **Ballot receipt.** Given that $\langle o, s \rangle$ were randomly selected in section 6.2.2, and known only to the privacy entrusted entity, then to all other entities $\langle s \rangle \not\Rightarrow \langle o \rangle$ and therefore $\langle s, \rho \rangle \not\Rightarrow \langle o, s, \rho \rangle$ given receipt $\langle s, \rho \rangle$.
- **Receipt Commitment Lists.** Given *ballot commitment lists*, $\mathcal{CB}_g = \{\langle c_{g_1}, o_1 \rangle, \dots, \langle c_{g_b}, o_b \rangle\}$ and $\mathcal{CB}_p = \{\langle c_{p_1}, o_1 \rangle, \dots, \langle c_{p_b}, o_b \rangle\}$ and *receipt commitment lists* $\mathcal{CR}_g = \{\langle c_{g_1}, s_1 \rangle, \dots, \langle c_{g_b}, s_b \rangle\}$ and $\mathcal{CR}_p = \{\langle c_{p_1}, s_1 \rangle, \dots, \langle c_{p_b}, s_b \rangle\}$, but given that only one of $\langle \mathcal{CB}_g, \mathcal{CR}_p \rangle$ and $\langle \mathcal{CR}_g, \mathcal{CB}_p \rangle$ are ever made public, it is

easy to see $\langle c_{g_i}, o_i \rangle \not\Leftarrow \langle c_{g_i}, s_i \rangle$ and $\langle c_{p_i}, o_i \rangle \not\Leftarrow \langle c_{p_i}, s_i \rangle$ and thus $\langle c_{g_i}, \rho_i \rangle \not\Leftarrow \langle o_i, s_i, \rho_i \rangle$ and likewise $\langle c_{p_i}, \rho_i \rangle \not\Leftarrow \langle o_i, s_i, \rho_i \rangle$.

6.4.1 Prevented Attacks

Under the privacy properties, a link cannot be established between a vote and a receipt, and therefore the receipt cannot be used to prove how a voter voted. That is to say any observer, given only the ballot receipt, cannot “guess” a voter’s selections with non-negligible advantage. These privacy properties address the following attacks:

- **Vote buying.** A voter who votes a certain way, following an arrangement made between the voter and any entity *a priori* to voting, cannot subsequently prove how they voted. This effectively relegates vote buying to, at best, conventional threats, or at worst, to the previously mentioned forced-randomization attack,
- **Retribution.** A subtly different threat, often overlooked in the literature, is the circumstance in which no precursory voting agreement exists, yet a possibility of retribution exists if the voter’s selections become known *a posteriori* to voting. Concern for future retribution would be legitimate cause for a voter to vote differently than intended, and therefore is arguably as serious a concern as vote buying and must likewise not be facilitated by any proposed receipt scheme.

6.5 Concluding Remarks

In this chapter we introduced Aperio, an election scheme with cryptographic-like verifiability based instead on physical tamper-evident paper documents. We believe Aperio may be useful as a tool for educating non-technical audiences about the concepts and procedures behind cryptographic election audits, without the comparatively conceptually-heavy demands of learning the cryptography itself. Through this approach, we believe Aperio can help foster the eventual acceptance of its cryptographic counterparts among the general public.

With Aperio, we deconstructed end-to-end election verification to its essentials. Nevertheless, relative to physical security assumptions, cryptographic assumptions offer stronger integrity/privacy assurances, in addition to being far more practical from the perspective of scalability. In the following chapter we bring Aperio into an electronic setting with the hope of maintaining a connection with its physical counterpart, while offering increased protections and scalability.

Chapter 7

Eperio

Is a system ‘universally verifiable’
when only a few mathematicians can
understand why it can be trusted?

David Dill [Dil07]

This chapter is adapted from published work co-authored with Jeremy Clark, and supervised by Urs Hengartner and Carlisle Adams [ECHA10].

7.1 Introductory Remarks

Aside from issues of the psychological/legal acceptability of cryptography in elections, there are questions about the technical complexity of performing the audits. We contend that the technical complexities added by cryptography are a barrier to widespread participation in the audits. For example, although the 2009 election in Takoma Park demonstrated that voters and election officials had a basic understanding of how to build and check receipts, only two individuals are known to have conducted the cryptographic component of the audit: both are experts in the field and worked in collaboration with us to implement their audit software.¹

In the interest of promoting wider participation in the election audit process, we believe it is important to minimize the role of cryptography in the audit procedures. Ideally, custom audit software could be avoided, or at least minimized.

¹Software and audit results available online at
<http://github.com/benadida/scantegrity-audit/> and
<http://zagorski.im.pwr.wroc.pl/scantegrity/>

In this chapter we present Eperio, an electronically tabulated version of Aperio. In its basic form the Eperio protocol involves a verifier downloading a set of encrypted files from an election website, opening a subset of them by entering a password and comparing the files to one another for consistency using a spreadsheet application or small software script.

Contributions. The contributions of this chapter are summarized as follows:

- The Eperio cryptographic election verification protocol,
- A proposal for implementing cryptographic commitments using symmetric-key encryption,
- A software implementation of the Eperio system, which includes,
 - An election verification script written in 50 lines of Python,
 - An election verification procedure using TrueCrypt and OpenOffice Calc spreadsheet application as an alternative to an automated script,
- A performance analysis of Eperio demonstrating that it requires fewer cryptographic operations, smaller audit datasets, less execution time and fewer lines of software than other recently deployed cryptographic election verification systems.

7.1.1 Motivation

Despite the important milestones made by previous work (cf. Section 2.2), cryptographic voting is often criticized, among other things, for being difficult to implement and conduct. The recent municipal election in the United States provides a good example: the two independent auditors of that election each wrote several hundred lines of software code to pore over the two and a half gigabytes of cryptographic audit data (supra note 1). Many systems in the literature employ advanced cryptographic primitives and techniques not typically found in standard software libraries. This often results in the cryptographic software components of such protocols needing to be custom coded, and has pointed us toward being less reliant on custom implementations.

Is simplicity central to the acceptance of cryptographic voting? On the one hand, there are examples of the general population accepting cryptography without understanding the protocols that enable it (e.g., online banking or wireless network privacy). It is also the case that some citizens do not fully appreciate current voting procedures (e.g., statistical recounts or ballot counterfoils) used to add some level of election integrity.

We contend that *universal verification*—the ability of anyone to participate in the election audit—is fundamental to the spirit of cryptographic election audits, and that lowering the technical complexity of the audits follows in that spirit.

Eperio Ballot and Receipt. As a research direction focused on improving the experience of election verifiers, Eperio is agnostic about the ballot and receipt mechanism. We only require that there exist some way to uniquely reference each optical-scan oval on each ballot in the election. We call this a unique markable region (UMR).

Without loss of generality, throughout the rest of this chapter we will use a randomized-candidate list ballot similar to the Prêt à Voter system [CRS05]. We note randomized candidate orderings are not legal in all jurisdictions, and that a randomized code ballot style (similar to Scantegrity) can be used interchangeably without modification to the Eperio protocol. To mark such a ballot, a voter first locates their preferred candidate and marks the associated optical-scan oval. The voter then separates the optical-scan ovals portion and candidate list portion, tearing along a perforation (see Figure 7.1). The ovals portion is scanned and retained by the voter as a receipt. The candidate list is shredded by the voter before leaving the polling place.

7.2 The Eperio Protocol

Eperio is closely related to the Punchscan and Scantegrity cryptographic election verification protocols. Unlike Punchscan and Scantegrity which use a mixnet-like structure to achieve the E2E integrity and privacy properties, Eperio combines its audit data into a single cryptographic table structure. This in turn permits a coarser, more efficient, cryptographic commitment scheme. It also facilitates interesting implementation options such as cryptographic commitments based on file-encryption and E2E verification in a spreadsheet.

As an intuition of this structure consider several ballot boxes, each of which contains a photocopy of each ballot cast in an election. If you shake one of the ballot boxes, the ballots will land with an ordering that is random and independent of the other boxes. However if you were to open that box and tally up the ballots, it will still produce the same winner as all the other boxes. In this way, verifying an election tally with Eperio constitutes proving that each such ballot box is a shuffled copy of (i.e., is *isomorphic* to) every other box. This physical analogy is inherited from Aperio. In the following sections, we will extend this approach to a cryptographic setting.

7.2.1 Protocol Sketch

As a brief overview of the protocol, a set of trustees will jointly generate a table with three columns. The first column will contain a unique reference for each markable region (*e.g.*, optical scan bubble) on each ballot in the election. The second column will indicate whether a given region was marked or not marked (or alternatively if the ballot was selected

for an audit). Finally the third column will contain the candidate/choice associated with each markable region.

The rows of this table are randomly shuffled (analogous to shaking a ballot box). It is easy to see that if the first two columns are revealed, the information should correspond to the set of all of the receipts in the election. If the last two columns are revealed, the information should correspond to the final tally. If all three columns are revealed (or the contents of an unrevealed column are implied through some functional dependency), then ballot secrecy is compromised. The Eperio protocol proves the correct formation of all three columns by only revealing the information implied in two of the three columns. It uses a composition of cut-and-choose and random audit techniques inspired by randomized partial checking [JJR02].

7.2.2 Entities

The Eperio protocol relies on the following entities, which are standard in most E2E voting systems:

- A set of n **election trustees** (or the prover), \mathcal{P} , tasked with generating a verifiable tally. \mathcal{P} is assumed to be a set of mutually distrustful and non-collusive trustees. However the protocol can tolerate $t \leq \frac{n-1}{2}$ trustees who collude or refuse to participate.
- The set of authenticated **voters** who cast ballots in the election.
- The first set of **verifiers**, \mathcal{V}_1 , who verify that their receipts were collected correctly (*Collected-as-Marked*). \mathcal{V}_1 are either voters, or auditors that were given access to a copy of a voter's receipt.
- The second set of **verifiers**, \mathcal{V}_2 , who verify that the ballots are printed correctly (*Marked-as-Intended*). \mathcal{V}_2 are either voters or auditors who went in person to obtain a ballot to audit.
- The third set of **verifiers**, \mathcal{V}_3 , who verify that the tally is computed correctly from the collected receipts (*Counted-as-Collected*). \mathcal{V}_3 can include anyone in any location with access to the election data.
- A malicious **adversarial prover**, \mathcal{P}' , who will attempt to convince the verifiers that an incorrect tally is correct.
- A malicious **adversarial verifier**, \mathcal{V}' , who will attempt to break voter privacy and determine which candidate was selected on a given receipt (or any non-negligible information about this selection).

7.2.3 Functions

The Eperio protocol requires a set of standard functions from the cryptographic literature: a threshold key agreement modeled after the one due to Pedersen [Ped91], a cryptographically secure pseudorandom number generator (PRNG), a perfect shuffle algorithm, a message commitment scheme (either perfectly binding or perfectly hiding), and a public coin (or random beacon) to generate non-interactive challenges.

Distributed Key Generation/Reconstruction

- $(y_1, \dots, y_n, \kappa) \leftarrow \text{DKG}(n, t, \ell, f_1, \dots, f_n)$
- $\kappa \leftarrow \text{KeyRec}(y_{j_1}, \dots, y_{j_{t+1}})$

DKG accepts from each of the n trustees a polynomial of degree t , f_i , with coefficients of bit-length ℓ such that they are elements of \mathbb{Z}_q for a suitably-sized prime q . The coefficients are summed together, mod q , to produce a new polynomial \hat{f} . Each trustee i receives $y_i = \hat{f}(i)$ as their share and the value $\kappa = \hat{f}(0)$ forms a shared secret of bit-length ℓ . t should be set such that $n \geq 2t + 1$. KeyRec accepts at least $t + 1$ shares, $y_{j_1}, \dots, y_{j_{t+1}}$, from a subset of the trustees and outputs the shared secret κ .

Pseudorandom Number Generation

- $\{0, 1\}^\ell \leftarrow \text{PRNG}(\kappa, \ell)$

PRNG is a stateful function which takes as input the shared secret, κ , as a seed and returns l new pseudo-random bits each time it is invoked. For simplicity, we omit ℓ if the size of the output is clear from the context.

Permutation

- $\pi(\mathbf{W}_i) \leftarrow \text{Permute}(w_1, \dots, w_u, \Pi)$
- $\pi(\mathbf{W}_i) \leftarrow \text{PermuteBlock}(w_1, \dots, w_u, s, \Pi)$

Permute accepts as input a list \mathbf{W}_i of u elements and a number, $\Pi = \mathcal{O}(u \log(u))$, of random bits sufficient to perfectly shuffle the list² and returns a list containing elements w_1, \dots, w_u in permuted order. For some s which divides u , PermuteBlock applies Permute independently to each of the u/s non-overlapping sub lists in \mathbf{W}_i .

²Generating a random integer from random bits is non-deterministic when the integer is not a perfect power of 2. Perfect shuffling algorithms, like Fisher-Yates, require random integers. An upper bound on the *expected* number of bits is $2(\log_2 u!)$.

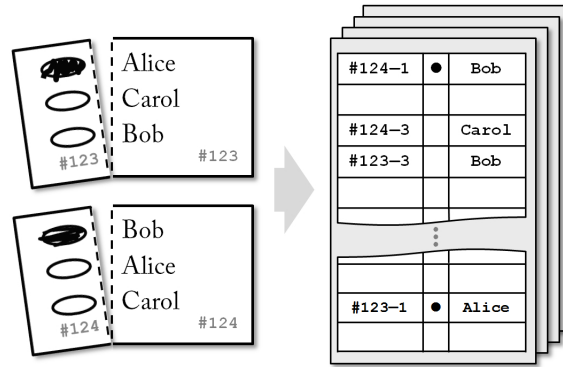


Figure 7.1: **Left: E2E-enabled optical scan ballots.** Each ballot consists of a unique serial number, a candidate list printed in an independent random order, and a perforation between the optical scan ovals and the candidate list. Upon marking the ballot, the candidate list is detached and shredded. The remaining piece is scanned and granted as a receipt. Because the candidate orderings are independent across ballots, knowing the mark position alone does not reveal how a voter voted. **Right: Eperio table.** Each optical scan oval (referenced by a serial number and absolute position), its mark-state (marked or unmarked) and the corresponding candidate name are recorded in a randomly assigned row.

Commitment

- $c \leftarrow \text{Commit}(m, r)$
- $\{0, 1\} \leftarrow \text{Reveal}(c, r, m)$

Commit takes as input an arbitrary length message m and a random factor r . It outputs a commitment to the message c . **Reveal** accepts values m, r , and c , and outputs 1 iff c is a valid commitment to m and r . Otherwise it outputs 0.

Public Coin Toss

- $\{0, 1\} \leftarrow \text{PublicCoin}()$

PublicCoin returns a uniformly random bit. The output should be unpredictable prior to being invoked and verifiable *a posteriori*.

7.2.4 Lists and Tables

The Eperio protocol relies on a particular data structure, called the Eperio table, which is constructed from a set of private inputs by the trustees. The Eperio table is a novel data

structure and the primary contribution of this chapter. It is shown in Figure 7.1 with a permutation-style ballot, which we use to illustrate the protocol.

To facilitate clarity, we also denote some intermediate lists and tables used in its construction. We use a bold typeface to denote ordered lists and tables, and a blackboard typeface to denote unordered sets.

The following list and set are public inputs to the system decided on prior to the election.

UMR List: \mathbf{U} is the list of each *unique markable region* (UMR) for the s candidates on each of the b ballots. Elements are encoded as a ballot serial number and a position, and are listed in ascending order. The length of the list is $u = s \cdot b$.

Candidate/Selection Roster: \mathbb{S} is the set of selections or candidates to appear on the ballot, for each contest. Elements are encoded as a character string of arbitrary length. The size of the set is s . Without loss of generality, we assume a single contest.

These are used by the trustees, in conjunction with the functions defined above, to create the following private list.

Candidate/Selection List: $\mathbf{S} \leftarrow \mathbf{U} \times \mathbb{S}$ is the list of candidates for each position on a ballot composed by randomly selecting, without replacement per ballot, an element from \mathbb{S} . It is ordered by \mathbf{U} . The length of the list is u .

The marks list will denote the final status of a markable region. It is empty prior to the election and is provided as a public input to the system after every ballot has been cast.

Marks List: \mathbf{M} is the list of marks corresponding to each markable region in \mathbf{U} . Elements include marked (1), unmarked (0), and print audited (-1). The length of the list is u .

The concatenation of these three lists defines a table that collects all the private information of the election.

Print Table: \mathbf{P} is a table formed by joining \mathbf{U} , \mathbf{M} , and \mathbf{S} . The dimensions of the table are $u \times 3$.

A proof of election integrity subsumes a proof that the relations between each list in \mathbf{P} is consistent with a universal view of the election. Pairwise, a correct $\mathbf{U-M}$ relation implies ballots were collected as marked (*Collected-as-Marked*), a correct $\mathbf{U-S}$ relation implies ballots were printed correctly (*Marked-as-Intended*), and a correct $\mathbf{M-S}$ relation implies the ballots were counted as collected (*Counted-as-Collected*). Note that revealing $\mathbf{P}_{(i,j)}$ for all i and j is sufficient, under our assumptions, for independently verifying the correctness of the tally. Unfortunately, this trivial approach would also destroy the privacy preserving property of the ballot receipt—in conjunction with \mathbf{P} , receipts would provide proof of which candidate was selected. Instead, we require a non-trivial approach that can both establish integrity and preserve ballot secrecy.

The Eperio table is a data structure that, with a set of queries, can prove the correct formation of \mathbf{P} while maintaining the same level of privacy provided by only revealing the list of receipts and the final tally. Specifically, it is a collection of x instances of \mathbf{P} that have been independently shuffled row-wise. By revealing portions of and relations on this structure, we will show that a complete and robust proof of integrity can be established with this minimal disclosure.

Eperio Table: \mathbf{E} is a table formed by x instances of \mathbf{P} , each of them independently shuffled row-wise. The dimensions of the table are $u \times 3 \times x$.

7.2.5 Protocol

We now outline the protocol for generating an Eperio table and the various outputs required for proving it encodes a correct tally. The focus of this chapter is on the protocol for verifying this proof, which is orthogonal to the issue of how the data is generated. However the security proof we provide in the next section encompasses the generation of the data, and so we give it consideration. The first three steps of the protocol are conducted prior to the election: initial setup, generating the Eperio table, and generating the commitments to the Eperio table. These steps are performed with a *blackbox computation* (for more on this primitive, see Section 7.4).

Initial Setup. The setup assumes that a list of candidates, \mathbb{S} , is available as well as the number of ballots, b , to be used in the election. The first task is for the trustees to generate an election secret and receive shares of this secret.

$$\circ (y_1, \dots, y_n, \kappa) \leftarrow \text{DKG}(n, t, l, f_1, \dots, f_n)$$

Generate Eperio Table. Next, the trustees generate the Eperio table \mathbf{E} . \mathbf{U} is formed by listing the ballot and position numbers in order. To form \mathbf{S} , the candidate list is repeated b times (which we denote by \mathbb{S}^b) and then randomly permuted on a ballot-by-ballot basis.

- $\Pi_S \leftarrow \text{PRNG}(\kappa)$
- $\mathbf{S}_i \leftarrow \text{PermuteBlock}(\mathbb{S}^b, s, \Pi_S)$

Table \mathbf{P} is created by placing \mathbf{U} , \mathbf{M} , and \mathbf{S} beside each other in columns. \mathbf{M} is initially empty, but in future meetings will include the marks recorded during the election. \mathbf{P} is used to print the ballots. We use the symbol $:$ to denote an entire vector within a matrix.

- $\mathbf{P}_{(:,1)} \leftarrow \mathbf{U}$
- $\mathbf{P}_{(:,2)} \leftarrow \mathbf{M}$
- $\mathbf{P}_{(:,3)} \leftarrow \mathbf{S}$

Finally, x independent row-wise shufflings of \mathbf{P} are generated. Each shuffled instance of \mathbf{P} is stored in the Eperio table, $\mathbf{E}_{(i,j,k)}$.

- For $k = 1$ to x ,
 - $\Pi_E \leftarrow \text{PRNG}(\kappa)$
 - $\mathbf{E}_{(:,1,k)} \leftarrow \text{Permute}(\mathbf{P}_{(:,1)}, \Pi_E)$
 - $\mathbf{E}_{(:,2,k)} \leftarrow \text{Permute}(\mathbf{P}_{(:,2)}, \Pi_E)$
 - $\mathbf{E}_{(:,3,k)} \leftarrow \text{Permute}(\mathbf{P}_{(:,3)}, \Pi_E)$

We define, for future use, the following function. It encapsulates all the steps in this ‘generate Eperio table’ section.

- $\mathbf{E} \leftarrow \text{GenE}(\kappa)$

Generate Commitments. The trustees are now ready to commit to the data in \mathbf{E} . For each instance $1 \leq k \leq x$, they will commit to both the $\mathbf{E}_{(:,1,k)}$ and $\mathbf{E}_{(:,3,k)}$ columns. This requires $x \times 2$ commitments. The random factors for these commitments are stored in an $x \times 2$ table \mathbf{R} . The resulting commitment is stored in the corresponding table \mathbf{C} .

- For $i = 1$ to 2 , $j = 2i - 1$, and $k = 1$ to x ,
 - $\mathbf{R}_{(i,k)} \leftarrow \text{PRNG}(\kappa)$
 - $\mathbf{C}_{(i,k)} \leftarrow \text{Commit}(\mathbf{E}_{(:,j,k)}, \mathbf{R}_{(i,k)})$
- Publish \mathbf{C}

Note that $2i - 1$ is simply the mapping $\{1 \rightarrow 1, 2 \rightarrow 3\}$, used to denote that commitments to the first and third columns of \mathbf{E} are stored, respectively, in the first and second rows of \mathbf{C} . \mathbf{C} is published to the bulletin board.

Voting. Registered and authenticated voters are issued a paper ballot with a randomized candidate list according to \mathbf{P} . After marking the ballot, the candidate list is detached and destroyed (*e.g.*, placed in to a paper shredder). The remaining strip is scanned by an optical scanner and the strip is retained by the voter as a receipt. The optical scanners will record for each ballot which position was marked, as well as the ballots that were print audited. After the election, these are placed into the list \mathbf{M} . Without knowing $\mathbf{P}_{(:,3)}$, this information does not reveal which candidate was voted for and can be published.

- **Publish \mathbf{M}**

Compute Tally. At least $t + 1$ trustees submit their election secrets to the blackbox computation, which regenerates the key and the Eperio table. This time, the completed marks list is shuffled along with the rest of the table.

- $\kappa \leftarrow \text{KeyRec}(y_{j_1}, \dots, y_{j_{t+1}})$
- $\mathbf{E} \leftarrow \text{GenE}(\kappa)$

For each instance x , they publish the corresponding marks list. Each of these lists is a shuffled version of the original \mathbf{M} .

- **Publish: $\mathbf{E}_{(:,2,:)}$**

Finally, a tally is computed from any $\mathbf{E}_{(:,2,k)}$ and $\mathbf{E}_{(:,3,k)}$ pair of columns, and the list of totaled values for each candidate, denoted τ , is published. This can be considered an *asserted tally*, as the purpose of E2E verification is to prove that this tally is correct.

Generate the Linkage List. To ensure that the Eperio table is consistent with what actually appears on the printed ballots in the election, verifiers have the ability to keep an issued ballot for purposes of auditing its printing. If a ballot was chosen to be print audited and the first position contained candidate Bob, then a row corresponding to this markable region will exist in \mathbf{E} at an unknown row. The row will be different for each instance. If the ballot is printed correctly, each corresponding row in each instance should contain Bob in the third column. The print auditor would like assurance of this fact.

However since commitments to only entire columns $\mathbf{E}_{(:,1,k)}$ and $\mathbf{E}_{(:,3,k)}$ exist, this fact cannot be directly revealed without revealing both columns for a given instance. Doing

this would reveal which candidate was selected for every receipt and cannot be pursued. Instead, the election trustees will indirectly establish this fact. The trustees assert the row number, a , in each instance corresponding to every audited markable region. Which markable regions are audited is contained in the marks list, \mathbf{M} , with -1 recorded for that entry. The list of asserted row numbers is called the linkage list, \mathbf{L} , and it is made public.

- for $i = 1$ to u and $k = 1$ to x :
 - if $\mathbf{M}_i = -1$:
 - **Find**: a s.t. $\mathbf{E}_{(a,1,k)} = \mathbf{U}_i$
 - $\mathbf{L}_{(i,k)} \leftarrow a$

Audit Challenge and Response. After the tally has been posted, the trustees prove to an independent auditor that the tally was calculated correctly. They do this through a cut-and-choose protocol. First the trustees regenerate the election secret and the Eperio table.

- $\kappa \leftarrow \text{KeyRec}(y_{j_1}, \dots, y_{j_{t+1}})$
- $\mathbf{E} \leftarrow \text{GenE}(\kappa)$

Next, they invoke the public coin toss function to generate one flip for each of the x instances.

- for $k = 1$ to x :
 - $z \leftarrow \text{PublicCoin}()$
 - $\mathbf{Z}_k \leftarrow z$
 - **Publish**: $\mathbf{R}_{(z+1,k)}$
 - **Publish**: $\mathbf{E}_{(:,2z+1,k)}$

Each flip is recorded in \mathbf{Z}_k . Depending on the flip, they either reveal the first two or last two columns in each instance. Recall that $\mathbf{E}_{(:,2,:)}$ was published previously. This is illustrated in Figure 7.2.

7.2.6 Verification

We now show the steps that the verifiers take to check that the published data corresponds to a tally that is correct. Recall there are three sets of audits (and corresponding verifiers). The first set, \mathcal{V}_1 , are the voters who check their receipts (or provide a copy to someone they delegate to check on their behalf). If the receipt corresponds to ballot number b and contains s positions that are either marked (1) or unmarked (0), the auditor should check

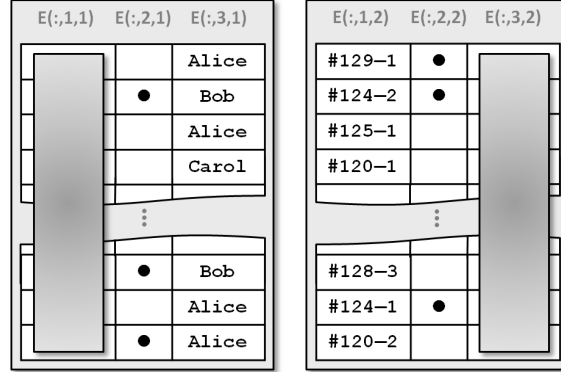


Figure 7.2: **Auditing Eperio table instances.** Two example instances of the Eperio table during auditing. Each instance alleges to contain the same information, but in an independently shuffled order. **Left:** $E(:,3,1)$ was challenged (then revealed), allowing verifiers to tally the election. **Right:** $E(:,1,2)$ was challenged (then revealed) allowing voters to check their receipts. The grey bars symbolize cryptographic commitments that will remain unopened to protect ballot secrecy.

that the status of each position i on the receipt matches the status recorded at \mathbf{M}_j , where $j = s(b-1) + i$.

The second set of verifiers, \mathcal{V}_2 , should check the linkage list against their print audited ballots. Let $a = \mathbf{L}_{(i,k)}$ for an i on their ballot and an instance k . Depending on the random coin for instance k , \mathcal{V}_2 should check that $\mathbf{E}_{(a,1,k)}$ matches the associated markable region on the ballot or $\mathbf{E}_{(a,3,k)}$ matches the associated candidate. They should do this for all i on their print audited ballot and each k in the election.

These two audits establish that the reported marks correspond to what appeared on voters' completed ballots and that what appears on the ballot corresponds to what is in the pre-committed Eperio table. The final step is to ensure that the asserted tally, τ , corresponds to the marks. Recall that for each of the x instances, a random coin was flipped to reveal value $z \in_r \{0, 1\}$. The third set of verifiers, \mathcal{V}_3 , should do the following.

- for $k = 1$ to x :
 - $z \leftarrow \mathbf{Z}_k$
 - Check $\text{Reveal}(\mathbf{C}_{(z+1,k)}, \mathbf{R}_{(z+1,k)}, \mathbf{E}_{(:,2z+1,k)})$
 - If $z = 0$:
 - Check $\{\mathbf{E}_{(:,1,k)}, \mathbf{E}_{(:,2,k)}\} \cong \{\mathbf{U}_i, \mathbf{M}_i\}$
 - If $z = 1$:
 - Check $\{\mathbf{E}_{(:,2,k)}, \mathbf{E}_{(:,3,k)}\} \cong \tau$

Here \cong means that the two pairs of tables are isomorphic—*i.e.*, they are a permuted representation of the same information.

7.3 Security

In this section, we summarize the main results of our security proof, which can be found in the full paper [ECHA12]. Given a transcript of the entire protocol, the asserted tally can be either accepted or rejected. If the transcript is correct and matches the asserted tally, the decision will always be to accept (completeness). If the asserted tally is not correct, the decision will be to reject with a high probability (soundness). Finally, the outputs do not provide any information that can be used by a computationally bounded adversary to determine any non-negligible information about which candidate was voted for by any voter (computational secrecy).

Let \mathcal{P} be an unbounded prover (the election authority) and \mathcal{V} be a PPT-bounded verifier. Either entity may employ a malicious strategy and we denote this with a prime (\mathcal{P}' , \mathcal{V}'). Recall that τ represents the asserted tally and let ρ be the asserted receipts.

Soundness. The soundness of Eperio relies on two assumptions:

1. The function $\text{Commit}(m, r)$ is perfectly binding. That is, for any m_1 such that $\text{Commit}(m_1, r_1) = c_1$, there does not exist any r_2 and $m_2 \neq m_1$ such that $\text{Reveal}(c_1, r_2, m_2) = 1$.
2. The invocation of function $\text{PublicCoin}()$ at the start of the post-election audit is perfectly unpredictable at the close of the election.

Let b'_r be the number of modified ballot receipts, and $0 \leq p_1 \leq 1$ represent the fraction of voters who conduct a receipt check. Let b'_p be the number of misprinted ballots, and $0 \leq p_2 \leq 1$ be the fraction of ballots that are print audited. Recall there are x instances in $\mathbf{E}_{(:, :, k)}$, for $1 \leq k \leq x$. Given the above assumptions hold, it is proven (in the full paper) that the probability of \mathcal{V} rejecting a malformed transcript from \mathcal{P}' is:

$$\Pr[\text{REJECT}_{\mathcal{P}', \mathcal{V}}] = \min[(1 - (1 - p_1)^{b'_r}), (1 - (1 - p_2)^{b'_p})(1 - \frac{1}{2^x})].$$

Computational Secrecy. The secrecy of Eperio relies on the following assumptions:

1. The number of colluding trustees is at most t .
2. At least one trustee submits to DKG an f_i drawn with uniform randomness from \mathbb{Z}_q^t .

3. All outputs are computed with a blackbox.
4. Any polynomial-sized output from $\text{PRNG}(\kappa)$ provides no non-negligible advantage to a PPT-bounded adversary in guessing either κ , the next bit in the output, or any unobserved previous bit.
5. The function $\text{Commit}(m, r)$ is semantically secure and computationally hiding. That is, given either $c_1 = \text{Commit}(m_1, r_1)$ or $c_2 = \text{Commit}(m_2, r_2)$ for any chosen m_1 and m_2 , a PPT-bounded adversary should have no non-negligible advantage in guessing which message was committed to.

Let ϵ_{A4} and ϵ_{A5} be the advantage specified in assumptions 4 and 5. Given all of the assumptions hold, we prove in the full paper [ECHA12] that the advantage of \mathcal{P}' recovering non-negligible information about any voter's selection given the full transcript as opposed to just the final tally is:

$$|\Pr[\text{RecoverSel}(\text{View}_{\mathcal{V}'}(\rho, \tau, \mathbf{C}, \mathbf{L}, \mathbf{E}_{(:,2,:)}), \mathbf{R}_z) = 1] - \Pr[\text{RecoverSel}(\text{View}_{\mathcal{V}'}(\tau)) = 1]| \leq \epsilon_{A4} + \epsilon_{A5}.$$

Additional Claims. In addition to the above proofs, we also show in the full paper that Eperio is complete and can be modified to have everlasting privacy (*i.e.*, \mathcal{V} is unbounded and \mathcal{P} is PPT-bounded).

7.4 Practical Primitives

In this section, we revisit a few of the cryptographic primitives needed in Eperio. In particular, we are interested in options that allow for useful deployment options.

Blackbox Computation The Eperio protocol requires the generation of the Eperio table to be done using a blackbox computation. While in theory, the task performed by the blackbox could be made into a multiparty computation (where only privacy is required as correctness is provided by Eperio), we instead propose the use of a semi-trusted computer. It is semi-trusted in the sense of only providing *private* evaluation of functions; the *correctness* can be determined through the audit. That said, mechanisms are provided to encourage correct evaluation. To this end, we assume disclosed source code for the functions to be evaluated is provided in advance and some attestation mechanism is available to ensure it is the same code running on the computer.

All tasks performed by the trustees can be accomplished by regenerating the Eperio table, and the regeneration of this table can be accomplished through a threshold of secret shares from the trustees. Therefore the computer should not have any persistent memory and its internal state should be purged after the outputs have been published. While this assumption may seem strong, in each of the recent occasions where end-to-end verifiable voting systems were used in real-life binding elections, semi-trusted computers were deployed: *e.g.*, Punchscan at the University of Ottawa [ECCP07a], Helios at Université Catholique de Louvain (for key generation from the description in their paper) [AMPQ09], and Scantegrity at Takoma Park, MD (cf. Chapter 5).

Cryptographic Commitment. We are interested in using symmetric-key file encryption as a commitment function for its speed, simplicity and widespread availability of software implementations. For Eperio, this means putting the columns of the Eperio table into individual files, encrypting them under a randomly chosen key, and posting the encryption as a commitment. To open the commitment, the encryption key is revealed and the file can be decrypted. While our implementation of Eperio can be easily modified to work with any standard commitment function, we use this approach to simplify the experience for voters who want to verify the proof for themselves. File encryption utilities are readily available, easy to use, and the commitment has message recovery.

Let \mathcal{E} be a pseudorandom permutation (PRP): $\{0, 1\}^k \times \{0, 1\}^m \rightarrow \{0, 1\}^m$. Let M be a message of L m -bit blocks and let IV be a random m -bit initialization vector. Define E to be the cipher block chaining (CBC) mode of encryption that encrypts $M = m_1, \dots, m_L$ under k -bit key K and m -bit IV . E is defined as: $c_0 = IV$ and $c_i = \mathcal{E}(K, c_{i-1} \oplus m_i)$. Assume M is exactly $L \cdot m$ bits long and $k = m$. Let D be its inverse decryption function applied to $C = c_0, \dots, c_L$ under K : $M = D(K, C)$.

Theorem 1: As defined, E is indistinguishable under a chosen plaintext attack (CPA) [BDJR97]

Conjecture 1: As defined, D behaves like a pseudorandom function with respect to collisions when C is held constant (note the difference from standard assumptions on M and C with a fixed K). That is, $M \leftarrow D_C(K)$ has random collisions for a fixed C and a variable K .

Theorem 2: Define a *commitment with message recovery* function as $(c, IV) \leftarrow \text{Commit}(M, K) = E(K, IV, M || f(M))$, where E is, as defined, CBC

mode with a pseudorandom permutation, and $f(M) = M\|M$ is a redundancy function.³ Define $M' \leftarrow \text{Reveal}(C, K) = D(K, C)$. M' is only accepted when M' has the correct form $M\|f(M)$ for some M . We show that such a commitment is computationally hiding under Theorem 1 and statistically binding under Conjecture 1.

We omit a proof of Theorem 2 here but it is included in the full version of the paper [ECHA12]. Because the commitment has message recovery, its `Reveal` function differs slightly from the commitment used earlier. We have demonstrated a very specific statistically hiding commitment function that can be constructed from a block cipher, assuming conjecture 1 holds. We model this ideal functionality in the real-world with AES-128-CBC in the next section.

Public Coin Toss. Voting systems often require the use of a public coin for the purposes of fairly implementing the cut-and-choose aspect of the audits. In the case of conventional voting, it is used to select precincts for manual recounts. Cordero *et al.* suggest a protocol using dice [CWD06]. Clark *et al.* note that dice outcomes are only observable by those in the room, and suggest a protocol for auditing E2E ballots using stock market prices [CEA07b], which has recently been given a more formal analysis [CH10]. This was suggested earlier by Waters *et al.* outside of the voting context [WJHF04a]. A further alternative is to use the Fiat-Shamir heuristic [FS86], which is secure in the random oracle model. However, a requirement for Fiat-Shamir is that the challenge space is large. In our case, the number of challenge bits is the same as the number of proof instances—for 10 or 20 instances, Fiat-Shamir proofs can be easily simulated under real-world assumptions and is thus unsuitable for our needs. Thus, we use the stock market protocol. The output from a statistically-sound PRNG is seeded with a random extraction of a pre-selected portfolio of closing stock prices. Evaluation of challenges occurs at least a full business day after the audit data has been committed to [CH10].

7.5 Implementation

Election Generation Tool. The hardware/OS platform design of the election generation software tool follows directly from previous systems such as Punchscan and Scantegrity whereby a diskless, stand-alone computer is booted from a Linux live-CD. The Eperio election generation software is then loaded via a USB-key after being certified by external experts. Election data is generated and written back on to the USB sticks, at which

³We thank and acknowledge Ronald L. Rivest for suggesting this redundancy function for creating a binding commitment from a block cipher. Any errors in the analysis are our own.

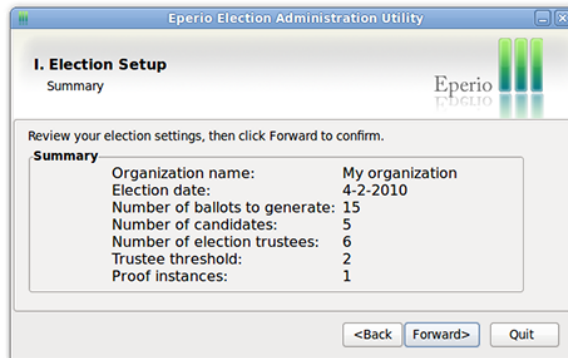


Figure 7.3: Election generation wizard. Trustees are guided through the complete process of selecting election parameters and setting up strong passwords prior to generating ballots and cryptographic audit data.

point the computer is shut down. The election generation software itself is separated into graphical user-interface, and cryptographic/back-end components. The primary cryptographic operations (i.e., file-encryption based commitments) were realized using function calls to OpenSSL. We implemented the election generation tool in Python using the GTK+ GUI library, which is a standard to most GNOME Linux live-CD distributions. The graphical layout consists of a multi-panel “wizard” work flow (shown in figure 7.3) to incrementally guide election trustees through parameter selection and password input. Unlike the Punchscan/Scantegrity software implementation, which requires some initial system configuration, the Eperio trustee interface software is intended to be self-contained and runnable directly following a live-CD boot. A specification of the audit data is given in Appendix A.1.

Bulletin Board. The implementation of secure public, append-only bulletin boards is an open area of research. In its simplest form, the bulletin board consists of signed election audit data available on a public FTP server administered by the election authority, and which is closely monitored and archived/mirrored by any interested party. Mirroring of audit data can be an important peer-service provided between verifiers since the election authority may maliciously attempt to modify commitment data throughout the course of the election. It also ensures long-term archiving of the audit data, which may otherwise be removed following the election.

Verification Script. As a primary objective of Eperio, the verification script was designed to be compact and execute swiftly. A Python script externally calls OpenSSL to decrypt relevant commitment files and then performs the audits on them. We implemented proofs of the *marked-as-intended* and *counted-as-collected* properties in a compact fifty (50)

lines of Python code (see Appendix A.2). As will be discussed in the following section, this represents the smallest implementation of a verification interface relative to other major released implementations by an order of magnitude. We tested this implementation on Ubuntu 7.10 (Python 2.5) and 9.10 (Python 2.6) as well as on Mac OSX Leopard and found that it could be executed without additional installation or configuration. Verifiers using Windows would be required to install Python and OpenSSL, or as an alternative, they could be directed to burn and boot Linux live CD. With the verification scripts on a USB key the entire audit can be completed without actually installing or configuring software on a verifier's machine.

Verification using Spreadsheets. An interesting alternative to writing a custom code-base is to use a spreadsheet that can import a CSV file as a worksheet. We believe that spreadsheets can help broaden the appeal of E2E verification, given many citizens who are not accustomed to reading/writing/running code *do* use spreadsheets. Spreadsheets have grown to become one of the most familiar computer applications, capturing a diverse cross-section of users in government, enterprise, and consumer sectors. They are also widely available: OpenOffice.org, a freely-downloadable open-source office productivity suite (which includes the Calc spreadsheet application), reports over a hundred million downloads.⁴

We have developed a set of simple manual verification steps using TrueCrypt and OpenOffice Calc. As an overview of this implementation, the user manually copies and pastes the revealed encryption keys into the TrueCrypt volume password dialog to mount the encrypted volume. The unencrypted CSV file can now be loaded directly in to the spreadsheet program as a worksheet. The verifier can then complete the audit using only simple spreadsheet operations (COPY, PASTE, SORT, etc). A full set of directions are given in Appendix A.3. As an alternative to manual verification, most spreadsheet applications integrate power macro/scripting languages that could automate the audit checks.

Spreadsheets as an All-in-one Election Audit Tool. Although many spreadsheets support basic file-encryption, cases of improper implementation (as in [Wu05]) have led to a general distrust of spreadsheets as an encryption service. We observed that OpenOffice's file encryption leaks partial information about a file's contents by storing an unencrypted/unsalted hash of the first 1024 bytes of the compressed spreadsheet,⁵ making it unsuitable as an implementation of the commitment scheme proposed in section 7.4. Indeed it is not clear whether any of the major contemporary spreadsheet applications would be suitable for implementing file-encryption based commitments. Nevertheless, the prospect

⁴<http://stats.openoffice.org>

⁵Open Document Format for Office Applications (OpenDocument) v1.2

of an “all-in-one” election verification tool already installed on most voters’ computers is an intriguing one.

7.6 Performance Comparison

Let us examine the technical requirements of election audits by comparing Eperio to several other major implementations. For space reasons we restrict our comparison to recent systems that have been deployed in binding elections, specifically Punchscan, Prêt à Voter, Helios and Scantegrity II. As outlined in Section 7.2, Eperio uses a cut-and-choose protocol to verify the correctness of the ballot data structure. Through the use of the *linkage-list* construct for print auditing, and the symmetric-key commitment scheme outlined in Section 7.4, the protocol allows for the commitment of entire columns, causing the number of required symmetric-key block operations to grow with the number of voters/candidates divided by the block size. The compactness of the verification process follows from it: verification involves running a file-decryption utility, followed by a sort and comparison of columns with the asserted outcome.

System	Mod Exp	Mod Mult	Sym Key Ops	Data (B)	LOC (approx)
Eperio	–	–	$3iv(\log(2v))/16$	$6iv(\log(2v) + 2) + 16i$	50
Scantegrity	–	–	$7.5civ$	$168civ$	1000
Punchscan	–	–	$7.5iv + 9v$	$168iv + 224v$	2000
Prêt-à-Voter	$1.5itv$	–	itv	$288itv + 192v$	1000
Helios v2.	$9cv + ct$	$9cv$	$2.5cv$	$2064cv$	1500

Table 7.1: Technical requirements for election verification: comparison of Eperio with other implementations for cryptographic audits involving v voters, c candidates, t trustees and i proof instances.

Both Punchscan and Scantegrity rely on a similar cut-and-choose protocol to audit their respective data structures, but utilize a particular commitment and print-audit scheme that requires each table element to be committed to separately resulting in a much larger dataset and number of symmetric-key operations. Unlike Eperio and Scantegrity however, Punchscan encodes ballot information in a way that is mostly invariant to the number of candidates, resulting in some savings in terms of data/number of overall block operations, although it requires additional ballot commitments as a result of its particular ballot style. Punchscan was first deployed during University of Ottawa Graduate Students Association’s (GSAÉD) annual general election in 2007 [ECCP07a]. This election dataset (denoted GSAED07 in Table 7.2) and verification source code (written in C#) are available online.⁶ Scantegrity was deployed during the 2009 Takoma Park municipal election (supra note 2).

⁶<http://punchscan.org/gsaed/>

This election dataset (denoted **TAKOMA09** in Table 7.2) and verification source code (Python and JAVA) are also available online (supra note 3). We used the Python implementation to produce timing results in Table 7.2.

Prêt à Voter [CRS05], also an E2E mechanism for optical-scan elections, uses randomized partial checking of the correct behaviour of nodes in a decryption mixnet [JJR02]. Each election trustee separately maintains two mix nodes in this network, and auditing the mixnet involves verifying the public-key decryption of half of the ciphertexts emanating from a given mix node. The number of public-key operations a verifier must perform therefore is the number of trustees times the number of voters. Auditing ballot printing also involves decryptions proportional to the number of trustees. Prêt à Voter differs from Eperio, Punchscan and Scantegrity in that it does not pre-commit to cryptographic ballot forms—they can be generated (and audited) on-demand. This means only cast ballots are included in the mixnet, but the print audit still requires more ballot forms be printed than will actually be cast. Prêt à Voter was deployed in 2007 for the University of Surrey Students’ Union (USSU) Sabbatical Elections. The audit dataset (**USSU07**) is not currently available online. The verification interface was written in JAVA.⁷

Helios v2. [AMPQ09] is an E2E mechanism for remote voting that uses homomorphic tallying (i.e., tallying under encryption) to protect voter privacy. To ensure the tally is correct, a zero-knowledge proof of correctness accompanies the encryption of each candidate on each ballot in the election. Likewise, a proof of decryption accompanies the final tally. The primary computational requirement of Helios election verification comes from auditing the inputs—each ballot requires 4 modular exponentiations per candidate. The Helios verification interface was written in Python.⁸ Helios v2. was deployed in 2009 in an election at Universite Catholique de Louvain. The the election dataset (**UCL09**) is not currently available online. Timing analysis of a reference election of the original implementation, Helios v1., is presented in [Adi08] and is denoted in Table 7.2 as **HELv1REF**.

Audit Dataset Size. For an election involving v voters, c candidates, t trustees and i proof instances, Table 7.1 expresses each system in terms of the number of modular exponentiations, modular multiplications, and symmetric-key block operations required during the verification process, along with the size of the audit dataset (bytes) and contributed software lines of code (LOC) of the respective implementation. In the case of paper-ballot systems, we include an additional $0.5v$ ballots for the print audit. In the case of systems with pre-committed ballots, an additional $0.5v$ ballots are included to account for spoilage. Although the execution and data complexities of the systems are all linear in the election parameters, Eperio’s performance advantage stems from its smaller constant factor.

⁷<http://www.pretavoter.com/>

⁸<http://www.heliosvoting.org/>

Election	Scantegrity	Punchscan	Helios	Eperio
TAKOMA09	1127s	–	–	162s
GSAED07	–	75s	–	9s
HELv1REF	–	–	14400s	1s

Table 7.2: Comparison of election verification times. Eperio times are for simulated datasets of equivalently sized election.

Timing. Basic timing analysis of election verification on available data is presented in Table 7.2. Our test platform was a 1.8GHz dual-core HP laptop running Ubuntu Linux 9.10. Eperio timings were performed on data *simulating* equivalently sized elections.

Comparing execution times of the Eperio verification script relative to three other systems (Punchscan, Scantegrity, Helios) on three election datasets shows Eperio significantly faster in time-to-verify. As a concrete example, using the Punchscan verification tool to audit the 2007 Punchscan election in [ECCP07a], it took us 75 seconds to complete, while an equivalently sized election run with Eperio took us 9 seconds.

The timing result for the Scantegrity audit on the TAKOMA09 dataset includes several audits relating to the Scantegrity II ballot and voting method of the specific election that we did not replicate, suggesting that a more precise timing analysis would produce a smaller margin.

Code Size: Is Less Really More? Table 7.1 lists the approximate code size of published verification software showing Eperio having a significantly smaller code base than related systems. As a performance metric however, software lines of code (LOC) is limited. There is debate as to how code lines should be counted, as well as which components to even include. It also does not take into account whether more efficient implementations could be created. Setting these questions aside, are small code bases important for E2E audit software?

While ultimately smaller code bases are not a proof of relative simplicity or ease of implementation, we believe it is a signal of simplicity as well as a gesture for engaging a wider audience.

7.7 Concluding Remarks

Despite fundamental limitations, source code reviews remain the conventional approach to electronic voting today. Such undertakings involve code bases of hundreds of thousands of lines of code that often may only be scrutinized by independent observers under strict

conditions. By contrast, cryptographic election verification is inherently public, universal, and undertaken on results themselves, not the machines that produce them. The software involved is also typically much smaller than a DRE. With Eperio, the software is smaller still—four orders of magnitude smaller—and verification can even be performed manually without any custom software. By making verification more accessible to voters, we contend that Eperio is an important democracy enhancing technology.

Eperio, like Scantegrity, relies on blackbox computation to protect ballot secrecy. Future work could explore a distributed version of the protocol, as well as a deeper exploration of the security of commitments based on block ciphers, such as proving its security in a suitable model (e.g., the ideal cipher model).

Chapter 8

CommitCoin

With cryptographic voting schemes, some of the most critical events occur very early in the election cycle, at a time when it is difficult to find observers.

Douglas W. Jones [Jon09]

This chapter is adapted from published work co-authored with Jeremy Clark [CE12].

8.1 Introductory Remarks

The election schemes presented in the previous chapters have, for the most part, been based on cryptographic commitments. The soundness of a commitment scheme, and hence the election itself, fundamentally cannot be assured unless relevant messages are exchanged in the proper order. A fundamental limitation of election schemes based on commitments, therefore, is an *early-participation* requirement placing a number of constraints on election verification:

- It requires a verifier to understand the principals of election verification (or at least of cryptographic commitments) prior to the election,
- It requires a verifier to be aware that the particular election will be offering the option of cryptographic verification,

- It requires a verifier to perform an action, i.e., download the commitments, prior to the election.

We believe requiring would-be verifiers to plan ahead for an election audit represents a significant obstacle. Central to our goal of making election verification more accessible, we seek to eliminate this early-participation requirement.

In this chapter, we show that an election authority can produce a commitment and later convince a verifier that the commitment was made prior to the election, premised on the assumption that the election authority does not have unusual computational power. We call this “carbon dating.” We show a general approach to carbon dating using moderately hard puzzles and then propose a specific instantiation. `CommitCoin` harnesses the existing processing power of the Bitcoin network without trusting it, and is designed to leave the commitment value evident in the public Bitcoin transcript in a way that does not destroy currency.

Contributions. The contributions of this chapter are summarized as follows:

- The carbon dating paradigm—a fuzzy time stamping mechanism based on moderately hard puzzles,
- `CommitCoin`, a protocol for carbon dating cryptographic commitments using the Bitcoin P2P network,
- An application of `CommitCoin` in carbon dating the Scantegrity commitments of the 2011 Takoma Park municipal election.

8.2 Preliminaries and Related Work

Secure Time-Stamping. Secure time-stamping [HS90] is a protocol for preserving the chronological order of events. Generally, messages are inserted into a hash chain to ensure their relative temporal ordering is preserved under knowledge of any subsequent value in the chain. The chain is constructed by a distributed **time-stamping service (TSS)** and values are broadcast to interested participants. Messages are typically batched into a group, using a hash tree [BdM91, BHS91, BLLV98, PRQ⁺98] or an accumulator [BdM93], before insertion in the chain. Time-stamping is a mature field with standardization¹ and commercial implementations.

¹ISO IEC 18014-3; IETF RFC 3161; ANSI ASC X9.95

A secure timeline is a “tamper-evident, temporally-ordered, append-only sequence” of events [MB02]. If an event E_{t_i} occurs at time t_i , a secure timeline can only establish that it was inserted after $E_{t_{i-1}}$ and before $E_{t_{i+1}}$ was. To determine t_i by consulting the chain, one must either trust the TSS to vouch for the correct time, or, to partially decide, trust a recipient of a subsequent value in the chain to vouch for when that value was received (if at t_j , we can establish $t_i < t_j$). However should conflicting values emerge, implying different hash chains, there is no inherent way to resolve which chain is correct beyond consensus.

Non-Interactive Time-Stamping. An approach closely related to our notion of carbon dating is **non-interactive time-stamping** [MSTS04]. In such a scheme, stampers are not required to send any message at stamping time. The proposed scheme is in the bounded storage model. At each time interval, a long random bitstring is broadcast to all parties. Stampers store a subset that is functionally dependent on the message they are timestamping. Verifiers also captured their own subset, called a sketch, at every time interval. This allows verification of the timestamp by anyone who is participating in the protocol, but not by a party external to the protocol. By contrast, our notion of carbon dating allows verification by anyone but is not necessarily non-interactive.

Proof of Work. The literature considers applications of moderately hard functions or puzzles that take a certain amount of computational resources to solve. These are variably called pricing [DN92], timing [FM97], delaying [GS98], or cost [GJMM98, Bac02] functions; and time-lock [RSW96, BN00, MMV11] or client [JB99, ANL00, DS01, WR03, WJHF04b, DMR06, TBFG07, CMSW09, SKR⁺11] puzzles. Proof of work is sometimes used as an umbrella term [JJ99]. Among other applications, proof of work can be used to deter junk email [DN92, GJMM98] and denial of service attacks [JB99, DS01, Bac02, WR03, WJHF04b], construct time-release encryption and commitments [RSW96, BN00], and mint coins in digital currencies [RS96, Bac02, Nak08].

We consider proof of work as three functions: $\langle \text{Gen}, \text{Solve}, \text{Verify} \rangle$. The generate function $p = \text{Gen}(d, r)$ takes difficulty parameter d and randomness r and generates puzzle p . The solve function $s = \text{Solve}(p)$ generates solution s from p . **Solve** is a moderately hard function to compute, where d provides an expectation on the number of CPU instructions or memory accesses needed to evaluate **Solve**. Finally, verification $\text{Verf}(p, s)$ accepts iff s is a correct solution to p .

Time-Stamping & Proof of Work. Bitcoin is a peer-to-peer digital currency that uses secure time-stamping to maintain a public transcript of every transaction [Nak08]. However new events (groups of transactions) are appended to the hash chain only if they include the solution to a moderately hard puzzle generated non-interactively from the

previous addition. Peers compete to solve each puzzle and the solver is awarded newly minted coins. A secure timeline with proof of work provides a mechanism to both limit the creation of new currency and to make it computationally difficult to change a past event and then catch up to the length of the original chain (peers accept the longest chain as canonical).

8.3 Commitments with Carbon Dating

A protocol for carbon dating commitments is provided in Protocol 8.1. It is a natural application of proof of work protocols but one that does not seem to have been specifically noted in the literature before.² Alice commits to a message m and instantiates a puzzle p based on the commitment value c that will take, on expectation, Δt units of time to solve. Alice begins solving p . Should a new party, Bob, become interested in when c was committed to, Alice will later produce the solution s . When given s , Bob concludes that p , and thus c , were created *at least* Δt time units before the present time. Since p will not take exactly Δt to solve, there is some variance in the implied instantiation time. We consider the case where Bob is only interested in whether the commitment was made well before a specific time of interest, which we call the **pivot time**.

PROTOCOL 8.1 (Commitments with Carbon Dating).

Input: Alice has message m at t_1 .

Output: Bob decides if m was known by Alice prior to pivot time t_2 .

The protocol:

1. PRE-INSTANTIATION: At t_0 , Alice commits to m with randomness r by computing $c = \text{Comm}(m, r)$. She then generates a puzzle based on c with difficulty d (such that the time to solve it is approximately Δt) by computing $p = \text{Gen}(d, c)$. She outputs $\langle c, p \rangle$.
2. INSTANTIATION: At t_1 , Alice begins computing $s = \text{Solve}(p)$.
3. RESOLUTION: At $t_3 = t_1 + \Delta t$, Alice completes $s = \text{Solve}(p)$ and outputs $\langle s, m, r \rangle$. Bob checks that both $\text{Verf}(s, \text{Gen}(d, c))$ and $\text{Open}(c, m, r)$ accept. If so, Bob decides if $t_3 - \Delta t \stackrel{?}{\ll} t_2$

If useful, a few extensions to Protocol 8.1 are possible. It should be apparent that carbon dating can be used for any type of sufficiently random message (*e.g.*, plaintexts, ciphertexts, signatures, *etc.*) by replacing c in $\text{Gen}(d, c)$ with the message. Second, the commitment can be guaranteed to have been made *after* a given time by, *e.g.*, including recent financial data in the puzzle instantiation [CH10]. Finally, the resolution period can

²Concurrent to the review of this work, it is independently proposed and studied [MVM11].

be extended by instantiating a new puzzle with the solution to the current puzzle (assuming the puzzles are entropy-preserving; see [GS98] for a definition of this property).³

8.3.1 Puzzle Properties

For carbon dating, we require the proof of work puzzle to have specific properties. Consider two representative proof of work puzzles from the literature (and recall c is the commitment value and d is a difficulty parameter). The first puzzle (P_{rs}), based on repeated squaring, is to compute $\text{Solve}(d, c, N) = c^{2^d} \bmod N$ where $N = q_1 q_2$ for *unknown* large primes q_1 and q_2 , and $2^d \gg N$ [RSW96, BN00, KC10]. The second puzzle (P_h), based on hash preimages, is to find an x such that $y = \mathcal{H}(c, x)$ has d leading zeros (where \mathcal{H} is a cryptographic hash function)⁴ [GJMM98, ANL00, Bac02, Nak08]. We contrast the properties of P_{rs} and P_h with the properties of an ideal puzzle scheme for carbon dating (P_{cd}).

P_{cd} should be moderately hard given a sufficiently random c as a parameter. P_{rs} requires d modular multiplications and P_h requires 2^{d-1} hashes on average. Neither precomputation, amortizing the cost of solving many puzzles, or parallelization should be useful for solving P_{cd} . Parallelization is useful in solving P_h , while P_{rs} is by design inherently sequential. Verify in P_{cd} should be efficient for anyone. This is the case in P_h but not P_{rs} , where efficient verification requires knowing the factorization of N ,⁵ making P_{rs} useful only when the puzzle creator and solver are *different* parties.⁶ When surveying the literature, we found that like P_{rs} and P_h , each type of puzzle is either parallelizable or only verifiable by the puzzle creator. Designing a non-interactive, non-parallelizable puzzle appears to be an open problem.

Finally, we require a few properties specific to our scheme. It should be hard to choose c such that the puzzle is not moderately hard. Given $s = \text{Solve}(\text{Gen}(d, c))$ and $s' = \text{Solve}(\text{Gen}(d, c'))$, it should be hard to find any pair of puzzles such that $s = s'$. Further, it should not be efficient to convert $\langle s, c \rangle$ into $\langle s', c' \rangle$.

³It may be preferable to solve a chain of short puzzles, rather than a single long puzzle, to allow (by the law of large numbers) the average solution time to converge and to reduce the amount of time Bob must wait for the solution.

⁴Let $\mathcal{H} : \{0, 1\}^* \rightarrow \{0, 1\}^m$. Then for $d \leq m$, find any x such that $y \in (\{0\}^d \parallel \{0, 1\}^{m-d})$.

⁵The totient of N serves as a trapdoor: compute $\delta = 2^d \bmod \phi(N)$ and then $s = c^\delta \bmod N$.

⁶Alice could use P_{rs} with the smallest unfactored N from the RSA challenges. Assuming continued interest in factoring these numbers, Alice's solution will eventually be verifiable. However she risks (a) it being factored before she solves the puzzle or (b) it never being factored at all. It also assumes non-collusion between Alice and RSA (assuming they know the factors).

8.3.2 Limitations

Aside from a good candidate for P_{cd} , the primary limitation to Protocol 8.1 is that the implied instantiation time is fuzzy. Carbon dating is best when the ratio between instantiation-to-pivot and pivot-to-resolution is maximized but the timing of the pivot is often unknowable. Another limitation is that Alice could commit to many different messages but only claim one. This excludes carbon dating (and non-interactive timestamping) from, *e.g.*, predicting election results or game outcomes. Generally, the scheme only works for accepting a committed message from an exponentially large set. A final limitation is that Alice must devote a CPU to solely solving the problem for a long period of time. We address this last limitation with **CommitCoin**, and then later provide an example where the first two limitations are not as applicable.

8.4 Carbon Dating with Bitcoin

Bitcoin is a peer-to-peer digital currency. A simplification of the scheme is as follows: participants are identified by a public signing key. A transaction includes a sender, receiver, and amount to be transferred (units of bitcoins are denoted BTC), and it is digitally signed by the sender and broadcast to the network. Transactions are batched together (into a “block”) and then appended to a hash chain (“block chain”) by solving the P_h hash puzzle on the block ($d = 53$ bits currently). The first node to broadcast a solution is awarded newly minted coins (currently 50 BTC) plus any transaction fees (currently optional). At the time of writing, one large Bitcoin mining pool, *Deepbit*, reports being able to compute 2^{42} hashes/second, while the network solves a puzzle on average every 10 minutes.⁷

8.4.1 The **CommitCoin** Protocol

If Alice can put her commitment value into a Bitcoin transaction, it will be included in the chain of puzzles and the network will provide carbon dating without Alice having to perform the computation herself. Bob only has to trust that Alice cannot produce a fraudulent block chain, longer than the canonical one and in less time. This idea has been considered on the Bitcointalk message board⁸ in the context of the distributed network vouching for the timestamp. Our observation is that even if you do not trust the timestamp or any node in the network, the proof of work itself can be used to carbon date the transaction (and thus commitment value).

⁷<http://deepbit.net>; <http://blockexplorer.com/q/interval>

⁸<http://goo.gl/fBNnA>

PROTOCOL 8.2 (CommitCoin).

Input: Alice has message m , key pair $\langle sk, pk \rangle$ associated with a Bitcoin account. Without loss of generality the account has a balance of > 2 BTC.

Output: The Bitcoin block chain visibly containing the commitment to m .

The protocol:

1. PRE-INSTANTIATION: At t_0 , Alice does the following:
 - (a) Alice commits to m with randomness r by computing $c = \text{Comm}(m, r)$.
 - (b) Alice generates new temporary key pair $\langle sk', pk' \rangle$ with $sk' = c$.
2. INSTANTIATION: At t_1 , Alice does the following:
 - (a) Alice generates transaction $\tau_1 = \langle pk \rightarrow pk', 2 \rangle$ to send 2 BTC from pk to pk' and signs it with randomness ρ : $\sigma_1 = \text{Sign}_{sk}(\tau_1, \rho)$. She outputs $\langle \tau_1, \sigma_1 \rangle$ to the Bitcoin network.
 - (b) Alice generates transaction $\tau_2 = \langle pk' \rightarrow pk, 1 \rangle$ to send 1 BTC from pk' back to pk and signs it with randomness ρ' : $\sigma_2 = \text{Sign}_{sk'}(\tau_2, \rho')$. She outputs $\langle \tau_2, \sigma_2 \rangle$ to the Bitcoin network.
3. TAG & OPEN: At t_2 , after τ_1 and τ_2 have been finalized, Alice generates transaction $\tau_3 = \langle pk' \rightarrow pk, 1 \rangle$ to send the remaining 1 BTC from pk' back to pk and signs it with *the same* randomness ρ' : $\sigma_3 = \text{Sign}_{sk'}(\tau_3, \rho')$. She outputs $\langle \tau_3, \sigma_3 \rangle$ to the Bitcoin network.
4. EXTRACTION: At t_3 , Bob can recover c by extracting sk' from σ_2 and σ_3 .

Remark: For simplicity we do not consider transaction fees.

Alice has control over at least three parameters in a Bitcoin transaction: her private key(s), her public key(s), and the randomness used in the signature algorithm which, importantly, is ECDSA. If she sets the receiver's public key⁹ to be her commitment value c and sends 1 BTC to it, the 1 BTC will be unrecoverable (akin to burning money). We consider this undesirable for two reasons: (a) it is financially wasteful for Alice and (b) it is not being a good citizen of the Bitcoin community.

By setting c equal to a private key or the signature randomness and following the protocol, c itself will never directly appear in the transcript. To get around this, Alice sets c to the private key of a new account and then purposely leaks the value of the private key by signing two different transactions with the same randomness. The CommitCoin protocol is given in Protocol 8.2. Since c is randomized, it has sufficient entropy to function (temporarily) as a secret key. A few bits of the secret key could be used as a pointer (*e.g.*, URL) to a place to post the opening of the commitment.

⁹Technically, it is a fingerprint of the public key.

8.5 A (Simplified) Example of CommitCoin

We committed to the title and abstract of the CommitCoin paper [CE12] and, as proof of concept, inserted it into the Bitcoin block chain on September 15, 2011 (the submission deadline for FC 2012). We used a simplified version of CommitCoin, with c set to be the public key fingerprint. As noted, it effectively burns a small amount of money. This example is intended as an easy-to-follow proof of concept for embedding values into the block chain using available tools. An implementation of Protocol 8.2 using c as the private key (avoiding such money burning) is left for future work.

First we ran,

```
openssl rand -out random.dat 20
```

creating a file containing a 20 byte random factor. Then we concatenated the randomness to the end of the abstract PDF and hashed it using RIPEMD-160,

```
cat abstract.pdf random.dat > preimage.dat
openssl dgst -ripemd160 preimage.dat
```

giving us the result:

```
135e3712334428d4061efe4e5ffd5ff817aeb817
```

This serves as a basic commitment scheme. We called an online tool¹⁰ to convert this hash into a valid Bitcoin address giving us:

```
12mQhvpvGYdBrvDJq6sFGwEe3GETaqEM4Jk
```

Finally, we used the Bitcoin Faucet¹¹ to send BTC0.005 to this address. This transaction ostensibly appeared in the Bitcoin blockchain on 2011-09-16 00:24:32 which can be seen in blockexplorer.¹² However it may be the case that we actually committed to our abstract long after 2011-09-16 00:24:32, and colluded with blockexplorer or the Bitcoin network to display the wrong timestamp. How can you really be sure?

¹⁰<http://blockexplorer.com/q/hashtoaddress/135e3712334428d4061efe4e5ffd5ff817aeb817>

¹¹<https://freebitcoins.appspot.com/>

¹²<http://blockexplorer.com/tx/2993c284f0b6838386ddd286af415384560d93cc4e27d25087fd6534f8e...>

At the time of publication, many more blocks have been added to the blockchain. Each block that is added is a solution to a moderately hard problem. Suppose we actually committed to the abstract yesterday. That means we would have had to forge the entire chain from Block 145535¹³ to the current block.¹⁴ Given each block takes on average 10 minutes for the entire Bitcoin network to solve, we would have required substantially more computing power than the entire Bitcoin network to have solved that many blocks in single day.

8.6 Use with Scantegrity

An interesting application of carbon dating is in cryptographic end-to-end verifiable elections such as Scantegrity. The soundness of such elections, however, relies in part on commitments made prior to the election. If a corrupt election authority changed the pre-election commitments after the election without being noticed, an incorrect tally could be made to verify. It is natural to assume that many people may only become interested in verifying an election after it is complete. Since the pivot (election day) is known, the commitments can be made well in advance, reducing the uncertainty of the carbon dating protocol. Moreover, owing to the design of Scantegrity, invalid commitments will only validate negligibly, ruling out precommitting to many possible values as an attack. Scantegrity was used in the 2011 municipal election in Takoma Park, MD (two years after the election in Chapter 5) and CommitCoin was used to provide carbon dating of the pre-election commitments. See Appendix B for details.

8.7 Concluding Remarks

Commitment-based election verification has the inherent limitation that verifiers are required to become involved in the audit process prior to the election. With CommitCoin, we can provide carbon dating of commitments in a way that is simple to verify, and through the use of BitCoin, nearly effortless to provide. In this scenario, even verifiers who become involved *a posteriori* to the election are able receive a proof that is sound from their view of the protocol transcript.

¹³<http://blockexplorer.com/b/145535>

¹⁴<http://blockexplorer.com/q/getblockcount>

Part III

Paper-ballot Based Elections with Distributed Trust

Chapter 9

Toward Oblivious Ballot Printing: A Two-party Approach

The printing press is either the greatest blessing or the greatest curse of modern times.

E. F. Schumacher

This chapter is adapted from published work co-authored with Jeremy Clark, and supervised by Urs Hengartner and Carlisle Adams [ECHA09].

9.1 Introductory Remarks

For the purposes of protecting ballot secrecy, a complete ballot should not be seen by any individual party except the voter. In the previous chapters, however, a single entity is entrusted with printing ballots. Toward the eventual goal of printing cryptographically verifiable optical-scan ballots in a distributed trust setting, we consider the problem of printing secret text (e.g., confirmation codes) on paper, in a human-readable format, without the printer(s) learning the result. In this chapter we propose a two-party protocol to distributively generate and print secrets. We leave integration with a verifiable voting system to following chapters.

Contributions. The contributions of this chapter are summarized as follows:

- A protocol for printing arbitrary-length messages using a trusted dealer,
- A protocol for two non-colluding printers to randomly and obliviously select and print a single element from a set of elements,
- A protocol for two non-colluding printers to randomly and obliviously select and print a random permutation on a set of elements,
- An optimization for printing alphanumeric characters using 16-segment display logic.

9.1.1 Visual Crypto Preliminaries

Recalling the description of visual cryptography from Section 2.1.7, we utilize a basic two-party version in this chapter. Consider a secret image s as an $m \times n$ matrix of pixels, where each pixel is either 0 for transparent or 1 for opaque. The first share α is generated by randomly selecting a 0 or 1 for each pixel. This share is then XORed with the secret image to generate the second share β . Thus, the original can be reconstructed by $s = \alpha \oplus \beta$. However, printing α and β on their own sheet of transparency paper and stacking them is equivalent to an OR operation, not an XOR.

To correct for this, the basic VC scheme maps each pixel in α and β into a 2×2 block of sub-pixels, which we call a VC-pixel. Without loss of generality this map is defined as $\square \rightarrow \begin{smallmatrix} \blacksquare & \square \\ \square & \blacksquare \end{smallmatrix}$ and $\blacksquare \rightarrow \begin{smallmatrix} \square & \blacksquare \\ \blacksquare & \square \end{smallmatrix}$. By layering VC pixels, we get either a fully opaque VC-pixel, defined as a 1, or a half-transparent VC-pixel, defined as a 0. This emulates the exclusive-or operation where: $\blacksquare\blacksquare = \begin{smallmatrix} \blacksquare & \square \\ \square & \blacksquare \end{smallmatrix} + \begin{smallmatrix} \square & \blacksquare \\ \blacksquare & \square \end{smallmatrix} = \begin{smallmatrix} \blacksquare & \blacksquare \\ \blacksquare & \blacksquare \end{smallmatrix}$, while $\square\square = \begin{smallmatrix} \blacksquare & \square \\ \square & \blacksquare \end{smallmatrix} + \begin{smallmatrix} \square & \blacksquare \\ \blacksquare & \square \end{smallmatrix}$ and $\square\blacksquare = \begin{smallmatrix} \blacksquare & \square \\ \blacksquare & \square \end{smallmatrix} + \begin{smallmatrix} \square & \blacksquare \\ \square & \blacksquare \end{smallmatrix}$.

9.2 Print an Arbitrary-length Secret Using a Dealer

General Model. A dealer D wants to have an arbitrary-length secret printed on a sheet of paper, intended for a recipient R , by a third party. D instructs two non-colluding entities offering print service, Printer A and Printer B, on how to print an image of a secret without either printer learning the secret.

Motivating Example. Consider the case where a bank, D , wants to distribute credit card numbers and activation codes to a large set of customers, R , through the mail. Due to the volume, the bank must outsource the printing to a printing service, Printer A, but is concerned that these secret numbers may be surreptitiously compromised. Instead, we would like to distribute the trust between two printers so that both printers would have to be compromised, or collude with each other, to learn the secrets.

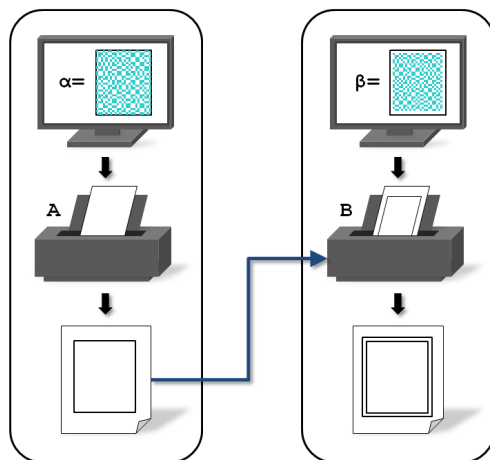


Figure 9.1: **Distributed printing.** Printers A and B receive separate visual crypto shares of a message. Printer A prints its share in invisible ink and passes the sheet on to Printer B who, in turn, prints its share on top.

Solution. The core of our proposed solution is a visual cryptography scheme in which individual shares are printed by separate, non-colluding, printers in succession onto a single sheet of paper using invisible ink, as shown in Figure 9.1.¹ This approach differs from the original VC proposal, which suggests printing shares on cellulose acetate (*i.e.*, overhead) transparencies, which can be aligned on top of a non-transparent share, such as a computer monitor or piece of paper, to reveal the secret [NS94]. The advantage of employing a single-sheet approach over that of a multi-sheet approach is threefold.

- **Alignment:** With VC, proper alignment (or registration) is necessary to reconstruct the message. Misalignments less than a sub-pixel reduce the contrast of the message, while greater misalignments render it unreadable [LWL09]. In our single-sheet scheme, alignment of the shares is a matter of attention for the printers, not the recipient. Presumably an industrial printing process is better suited to guarantee proper share alignment than the recipient.
- **Usability:** Our single-sheet approach offers a simpler user experience. The role of visual cryptography is not central to the recipient recovering the message and arguably, the recipient could be completely unaware of it.²
- **Fewer Chains of Custody:** Perhaps most important, physical separation of shares need not be enforced prior to the recipient receiving them. Assuming the scheme

¹While the last printer could print its share in non-invisible ink, we assume that each share is printed in invisible ink. This prevents the last printer's share from being learned if the paper is observed by earlier printers after being fully printed.

²Our scheme still requires the recipient to have a decoder pen, although in the case of trustworthy voting, the decoder pen directly substitutes the use of a normal pen when marking a ballot.

is secure within the privacy model described below, then the recipient will receive decisive feedback (i.e., tamper evidence) if the secret was viewed prior to its receipt.

As the primary application of invisible inks are in the transport of secret messages, in general we consider two threats to message secrecy:

- **Passive exposure:** A message written in invisible ink becomes temporarily visible in a particular environment (*e.g.*, ultraviolet light), to an optical sensor (*e.g.*, eye, camera, *etc.*).
- **Active exposure:** A message written in invisible ink is rendered permanently visible by initiating a one-way chemical process that develops (*i.e.*, activates) pigmentation in the ink.

We seek to mitigate these threats by requiring an invisible ink printing process with the following characteristics:

- **Indistinguishability of undeveloped ink:** A message printed in invisible ink is said to be resistant to passive exposure if any two messages are indistinguishable.
- **Tamper evidence:** A message printed in invisible ink is said to be resistant to active exposure if any actively attacked message is easily distinguished from an untouched message.

A simple attack for eschewing indistinguishability and tamper-evidence is one where a malicious party actively exposes (*i.e.*, develops) the message, records it, and reprints it on a new sheet of paper. The key to preventing this line of attack is in establishing the authenticity of the paper sheet. Using a document authentication scheme such as that mentioned in Section 2, either the dealer, recipient, and/or the printers would seek to establish document authenticity at some time after the printing process.

In the working example above, the bank could keep an inventory of sheets issued to Printer A and could verify their authenticity after having the sheets returned by Printer B (it could also at this time reveal the invisible ink). Alternatively, the recipient could perform the verification upon receiving the paper. In the following sections concerning schemes that do not employ a dealer, Printer A will publish an inventory of the sheets issued to Printer B. Both printers can check the sheets against the inventory at any point in the protocol, and after Printer B has applied its shares, the sheets can be authenticated by the recipient. A final alternative is to use an honest-but-curious third party to provide this service. In all cases, the verification could be conducted through a random audit of a small portion of the sheets.

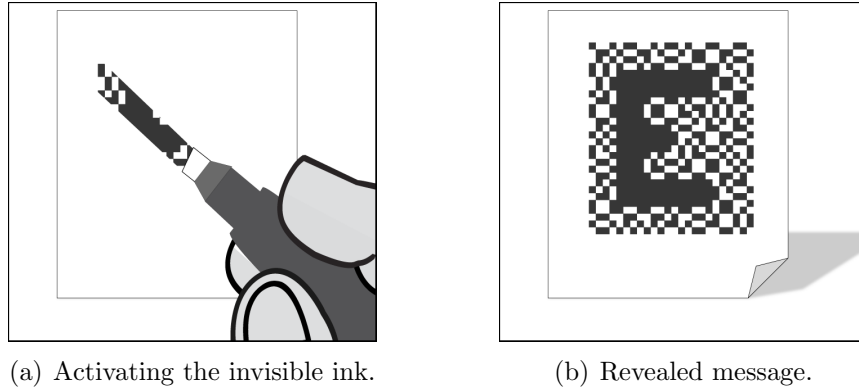


Figure 9.2: An obviously printed document from the recipient's perspective.

Definition 9.1. *Invisible Ink Secrecy Model.* We say a message printed in invisible ink is physically secure in the invisible ink secrecy model if that message is indistinguishable under passive exposure, and tamper-evident under active exposure.

Assuming the existence of an invisible ink printing system, secure in the invisible ink secrecy model, we propose a simple distributed printing scheme, outlined in algorithm 9.1, involving a trusted dealer D issuing two VC-shares of a secret to two printers: Printer A and Printer B. Printer A prints its share on a piece of paper in invisible ink. This sheet is given to Printer B, which prints its share in invisible ink, directly on top of Printer A's share. The resulting sheet is provided to a recipient R , who uses a special developing pen to reveal the combined shares (and hence the secret). See Figure 9.2.

9.3 Print a Randomly Selected Secret without a Dealer

General Model. Two non-colluding entities offering print service, Printer A and Printer B, randomly and obviously select a message from a public set of possible messages. The pixel representation (*i.e.*, image) of the message is printed on a sheet of paper intended for a recipient R without the use of a dealer. The selected message is unknown to both Printer A and Printer B and will only be known by R . Furthermore, printers A and B can each enforce the randomness of the selection independent of each other.

Motivating Example. Consider the case where Printer A and Printer B want to print a random alpha-numeric character on a sheet of paper to give to R . By doing this successive

Algorithm 9.1: Printing a secret with a trusted dealer

Dealer D's Private Input: Secret message $s \in [0, 1]^{m \times n}$ as $m \times n$ monochrome pixel matrix

1 Dealer D should:

- 2 | Fingerprint sheet of paper p
- 3 | Choose $\alpha \in_R [0, 1]^{m \times n}$
- 4 | Compute $\beta = s \oplus \alpha$
- 5 | Send α and β to Printer A and B respectively

6 Printer A should:

- 7 | Print α in invisible ink onto p
- 8 | Send p to Printer B

9 Printer B should:

- 10 | Print β in invisible ink on top of α on p
- 11 | Send p to Dealer D

12 Dealer D should:

- 13 | Authenticate paper p
 - 14 | Send p to recipient R
-

times, they could print a multi-character string where each character is independently selected at random. Scantegrity II, as an example, requires secret and random codes to be printed beside each candidate and already uses invisible ink to hide the value of the codes. Another example is ThreeBallot [RS07], which requires a set of statistically unique identifiers to be printed on each ballot. Printers or poll-workers who observe these identifiers, prior to the ballot being cast, threaten ballot secrecy.

Protocol. Let the set of possible messages be S and of order $N = |S|$. Consider, as in the motivating example above, that $S = \{A \dots Z, 0, \dots, 9\}$ in which each $s_i \in S$ shall be represented as an m by n monochrome pixel matrix that visually expresses it. At a high-level, Printer A will generate a random visual crypto image, α , as her share, and print it onto the paper in invisible ink. She will then, for each message $s_i \in S$, generate the complementary set of shares for Printer B: $\beta_i = \alpha \oplus s_i$, for $1 \leq i \leq N$. She randomly permutes the order of the set, obfuscates each element, and sends the set to Printer B. Printer B then selects $\gamma \in_R [1, N]$ to be deobfuscated, where \in_R denotes the uniform random selection of an element from a set. He then prints β_γ over top of α in invisible ink.

We require three additional properties of this distributed protocol in the absence of a dealer, namely:

- i. Printer A should not learn which visual crypto share β_γ was selected by B.

Algorithm 9.2: Printing a single secret character with oblivious transfer

Public Parameters: Set of alphanumeric characters S , and primitive roots $g, h \in \mathbb{G}_q$

```
1 Printer B should:
2   Choose index to select:  $\gamma \in_R [1, N]$ 
3   Choose random secret:  $x \in_R \mathbb{Z}_q^*$ 
4   Commit to private choice:  $y = g^x h^\gamma$ 
5   Send  $y$  to Printer A

6 Printer A should:
7   Perform lines 2, 3, and 7 from Algorithm 9.1
8   for  $1 \leq i \leq N$  do
9     Select  $i^{\text{th}}$  message from set:  $s_i \in S$ 
10    Compute complimentary share:  $\beta_i = \alpha \oplus s_i$ 
11    Randomly permute index:  $\beta_i \rightarrow \beta_{j=\pi(i)}$ 
12    Choose random secret:  $r_j \in_R \mathbb{Z}_q^*$ 
13    Compute:  $c_j = \langle a_j, b_j \rangle = \langle g^{r_j}, \beta_j (\frac{y}{h^{r_j}})^{r_j} \rangle$ 
14    Send  $p$  and set of  $c_j$ 's, ordered by  $j$ , to Printer B

15 Printer B should:
16   Flip coin  $c = \{H, T\}$  with  $\Pr[H] = \rho$ 
17   if  $c = H$  then
18     Request  $\alpha, \beta_j$ , and  $r_j, \forall j$ , from Printer A.
19     If correct, repeat protocol from Line 7.
20   else
21     Select  $c_\gamma$ 
22     Compute:  $\beta_\gamma = \frac{b_\gamma}{(a_\gamma)^x}$ 
23     Print:  $\beta_\gamma$  on top of  $\alpha$  on paper  $p$ 

24 Recipient or Printer A should:
25   Authenticate paper  $p$ 
```

- ii. Printer B should not learn the value of any share β_i other than the single share, β_γ , he selected.
- iii. Printer B should not know which message s_i corresponds to β_γ .

We utilize a 1-out-of- N oblivious transfer protocol to achieve properties (i) and (ii). s_i is perfectly hidden by Printer A's share α . Thus (iii) holds under the assumption of non-collusion of the printers.

The full details of the protocol are provided in Algorithm 9.2. The oblivious transfer sub-protocol is due to Tzeng [Tze04], which we selected for its reusable public parameters and minimal message exchange (2-pass). It is set in the ring of integers modulo a large prime p , with multiplicative subgroup of prime order q . By using a Pedersen commitment in line 4, Printer B's choice of share is perfectly hidden ensuring (i). Property (ii) holds

because line 22 for an arbitrary j reduces to $\beta_j h^{r_j(\gamma-j)}$ allowing the recovery of β_j only when $\gamma = j$.³ Property (iii) holds because $s_i = s_{\pi^{-1}(j)} = \beta_j \oplus \alpha$, and Printer B does not know α or random permutation $\pi()$.

Printer A could misconstrue the set of β_j values such that when they are combined with α , they do not each produce a unique character (*e.g.*, any selection by Printer B will result in the same character being printed). Given property (ii), Printer B could not detect such an attack directly. Thus Printer B shall, with some probability ρ , perform a cut-and-choose audit of Printer A's construction of α and β_j values to ensure they are properly formed.

An additional feature of the protocol is that both parties contribute to the *random* selection of $s_i \in S$. Printer A chooses a random permutation $\pi : i \rightarrow j$, and Printer B selects a random index γ to print. Thus if one of the two parties select messages deterministically, the contribution of the other will be to ensure the printed message is, in fact, randomly selected.

This protocol outlines how to print a single, secret, random, and obviously selected alphanumeric character on a piece of paper. It is easy to see that the oblivious transfer could be conducted several times, independently, to produce a string of characters for applications such as those offered above as motivating examples.

9.4 Print a Permutation of a Set of Messages without a Dealer

General Model. Two non-colluding entities offering print service, Printer A and Printer B, randomly select a permutation and apply it to a public set of possible messages. The pixel representations (*i.e.*, images) are printed on a sheet of paper intended for a recipient R without the use of a dealer. The order of these images is unknown to both Printer A and Printer B and will only be known by R. Furthermore, printers A and B can each enforce the randomness of the permutation independent of each other.

Motivating Example. A number of voting systems with cryptographic end-to-end integrity require ballots to be printed with a randomized candidate ordering. Prêt-à-voter

³Security remark: The only way for Printer B to recover β_j given $\langle g, h, x, \gamma, a_j = g^{r_j}, b_j = \beta_j h^{r_j(\gamma-j)} \rangle$ for $j \neq \gamma$ would be to perform a discrete logarithm: either solving $\log_g(g^{r_j})$ allowing B to compute h^{r_j} and then recover β_j by trying all values of $b_j / (h^{r_j(\gamma-j)})$, assuming a correct β_j is recognizable from some discernible structure, or alternatively, compute $\log_g(h)$ to find x' such that $g^{x'} h^j = y$ to similarly recover β_j from b_j . Concerning the latter, it is thus important that g, h are generated by Printer A or through a distributed key generation (DKG) protocol.

[CRS05] and Aperio [ECA08] require candidate names to be listed on the ballot in an independent random order. The candidate list could be developed immediately prior to voting to increase voter privacy. Alternatively, consider a contest where the names of prizes are printed in invisible ink on a set of tickets. For example, a batch of one dozen tickets could be printed as follows: ten shall say “please play again” while the other two shall each name a different prize. By applying a random permutation to these twelve strings, not even those running the contest will know for certain which tickets contain a prize.⁴

Solution. A permutation of a set of N images requires N elements to be printed. In its most basic form the 1-out-of- N oblivious transfer (see algorithm 9.2) is run N times, with Printer A selecting N independent, random, visual crypto shares α . However instead of applying independently selected permutations π_1, \dots, π_N at each successive execution, the same random permutation π_1 is applied to S when constructing the complementary set of VC shares $\{\beta_{(1,k)}, \dots, \beta_{(N,k)}\}$ during the k -th execution.

However since a permutation of elements requires every element to be appear once and only once, we shall require a mechanism to enforce non-repetition of elements. Such non-repetition of VC shares constructed by Printer A can be made through a similar cut-and-choose process as that mentioned in section 9.3. However to enforce non-repetition of the selections made by Printer B, we extend algorithm 9.2 by algorithm 9.3 such that Printer B proves the uniqueness of her selections, γ_k , without revealing the order of selection, as well as providing Printer A with the ability to perform a cut-and-choose on Printer B’s selections.

As a brief description of algorithm 9.3, Printer B constructs N commitments $y_k = g^{x_k} h^{\gamma_k}$ and sends them, along with the sum of random factors \hat{x} to Printer A. We index each message in the set using a public set of indices, selected from a superincreasing sequence κ (e.g., $\{k \in \mathbb{Z} : \kappa_k = 2^k\}$). Thus, if Printer B selects the same index more than once, it is impossible to adjust the other selections such that they sum to $\hat{\kappa}$ and are valid indices. Printer B could select an index not in κ , however this would forfeit him from learning at least one proper share. This will be caught by the cut-and-choose subprotocol in lines 12-15, which can be thought of as an optional step for scenarios where a malicious printer could get away with misprinting a share.⁵ To verify that each of the commitments y_k contains a unique index choice, Printer A will calculate their product \hat{y} causing the exponents γ_k to sum. Given public parameter $\hat{\kappa}$ and Printer B’s assertion \hat{x} , Printer A will verify whether $\hat{y} = g^{\hat{x}} h^{\hat{\kappa}}$ thus verifying B’s honesty in selecting each element once.

⁴This could prevent documented cases of fraud, such as, <http://archives.cnn.com/2001/LAW/08/21/monopoly.arrests/>

⁵For example, in the voting scenario, the permutation will be observed by the voter to be correct before it is used, thus not requiring this optional step. However it may be desirable for printing tickets in a contest.

Algorithm 9.3: Printing a secret permutation with oblivious transfer

Public Parameters: Superincreasing indices $\kappa = \{\kappa_1, \dots, \kappa_N\}$ that sum to $\hat{\kappa}$.

```
1 Printer B should:
2   for  $1 \leq k \leq N$  do
3     Choose, without replacement, index:  $\gamma_k \in_R \kappa$ 
4     Choose random secret:  $x_k \in_R \mathbb{Z}_q^*$ 
5     Commit to private choice:  $y_k = g^{x_k} h^{\gamma_k}$ 
6     Send  $y_k$  to Printer A
7   Compute  $\hat{x} = \sum_{k=1}^N x_k$  and send to Printer A

8 Printer A should:
9   Compute:  $\hat{y} = \prod_{k=1}^N y_k$ 
10  Verify:  $\hat{y} = g^{\hat{x}} h^{\hat{\kappa}}$ 
11  Flip coin  $c = \{H, T\}$  with  $\Pr[H] = \rho$ 
12  if  $c = H$  then
13    Request  $\gamma_k, x_k,$  and  $y_k, \forall k,$  from Printer B.
14    If correct, repeat protocol from Line 1.

15 for  $1 \leq k \leq N$  do
16   Run Algorithm 9.2 at line 7.
```

9.5 Efficiently Print Text

General Model. Two non-colluding printers, Printer A and Printer B, randomly select a short string of alphanumeric characters from a public set of possible strings. The model has the same properties as the general model in Section 9.3 but is optimized for the use of alphanumeric strings. It can be used in conjunction with Algorithms 9.2 and 9.3.

Motivating Example. An official is to issue a survey that contains a question about sensitive information that respondents may not answer honestly for fear of retribution. A mitigating technique is randomized response, where a sensitive question can be replaced with its negation in a random fraction of the surveys [AJL04]. Thus the issuers do not learn any particular respondent's true response with certainty, but can statistically adjust the results to estimate the number of respondents who answered in a particular way. Consider the problem of obviously printing a survey, where a particular question is randomly selected from a set that contains nine elements of question Q and one instance of question $\neg Q$. The answers are returned on a different sheet of paper (*e.g.*, a Scantron form), and the question sheet is discarded by the respondent after using it. The mechanism in this section can also be used to optimize the motivating examples in Section 9.4: for printing candidate names in voting systems or the names of prizes in a contest.

Solution. There is an upper limit to the size of the message that can be transferred in one instance of the protocol: in this case, it is the security parameter of the system, which is likely to be 1024 or 2048 bits. For the purpose of efficiency, our motivation is to encode as many characters as possible into one ciphertext payload.

We defined messages to be a monochrome pixel matrix of dimensions $m \times n$. Consider, for example, the image of a character to be 26 by 18 sub-pixels that are either white or black (the resolution that will be used in Figure 9.3). These parameters would require 117 bits per character, allowing just over half a dozen characters to fit into a single ciphertext payload. Increasing the resolution (and hence the readability) comes at the cost of using additional encryptions to convey the same content. To improve on this, we can use segment displays. In a segment display, alpha-numeric characters are displayed by driving a subset of 16 segments, or 7 segments if we restrict ourselves to numbers. At 16 bits per character, we can fit over 60 alpha-numeric characters into one 1024-bit secret: enough to encode a short question, candidate name—regardless of the character resolution.

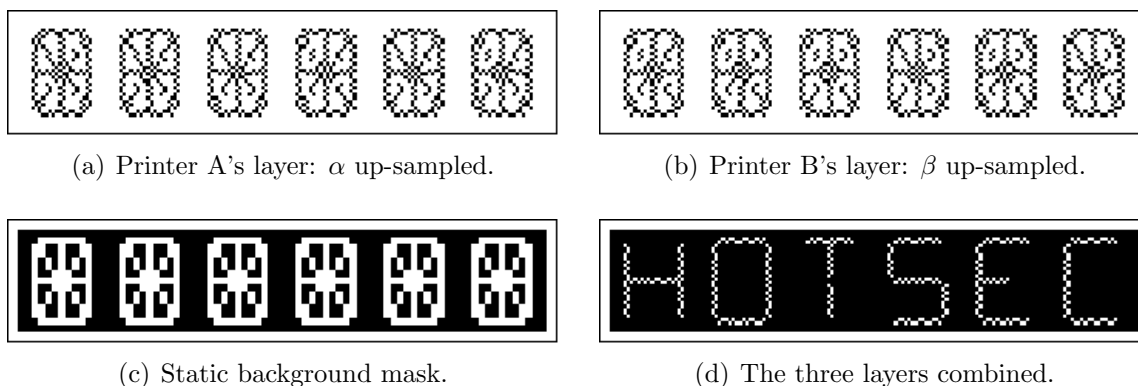


Figure 9.3: **Visual Crypto Up-sampling.** Example of a string of text using 16-segment up-sampling, comprised of two visual crypto layers set against a static background. Instead of representing each segment as a collection of traditional 2×2 visual crypto pixels, each segment's share can be transmitted as a single bit and then 'up-sampled' to an arbitrarily large, public, and pre-arranged visual crypto pixel pattern for improved perceptual clarity. In this way visual crypto shares can be fully expressed by 16 bits per alphanumeric character, regardless of perceptual resolution.

With the use of Algorithm 9.2, Printer A can obviously transfer one of many message shares to Printer B, where the share (β_γ) is a sequence of 16-bit segment encodings. If the exclusive-or of this sequence is taken with Printer A's accompanying sequence (α), the result is the character encoding of a randomly selected string (s_γ). However, we deviate from Algorithm 9.2 in that Printer A and B cannot print α and β_γ directly. Instead Printer A and B must up-sample their 16-segment sequences into a VC share.

The process of up-sampling works as follows. A pixel matrix of any dimension is partitioned into segments and background; white and black in Figure 9.3(c) respectively.

To aid with perception, the segment portion (white portion) is filled with random-looking VC pixels (recall a VC pixel is a 2x2 sub-pixel representation of a pixel). This same pixel mask can be used for every character. Printer A up-samples α by taking the pixel mask and for every segment, either leaving the segment as is if α is 0 for that segment or flipping all the bits in the segment if α is 1. See Figure 9.3(a). Printer A prints this and the background mask in invisible ink. Similarly Printer B up-samples β using the same pixel masks, generates a VC share in Figure 9.3(b), and prints it.

9.6 Concluding Remarks

We have demonstrated an interesting, novel paradigm: obviously chosen messages can be printed on a sheet of paper without the printers learning the result. We have outlined a number of scenarios where such a property may be useful, including password distribution, cryptographic voting, contests, and randomized response surveys. Indeed these protocols may be relevant to other applications of visual cryptography, in particular those requiring a dealer-less solution, as well as to other types of document and display media.

In particular, we pursue this line of work as a starting point for distributing the association between confirmation code and associated candidate on cryptographically verifiable optical-scan ballots—a necessity for strong ballot secrecy. The methods presented in this chapter could be used to print a randomized confirmation code on a ballot in a two-party distribution of trust. Integrating the outcome into the broader cryptographic election protocol, however, is not immediately possible. In the following chapter, we will integrate the techniques presented in this chapter into a Scantegrity-like voting system with distributed trust.

Chapter 10

Scantegrity 3-D: Scantegrity with Distributed Trust

Every officer, clerk, and agent in attendance at a polling station shall maintain and aid in maintaining the secrecy of the voting.

Ballot Act, 1872. United Kingdom

This chapter is adapted from published work co-authored with Christian Henrich and Urs Hengartner [EHH11].

10.1 Introductory Remarks

Despite the usability advantages of a paper optical-scan ballot, the cryptographic schemes presented in the previous chapters actually offer less protection to ballot secrecy than conventional elections do. In Scantegrity, for example, knowing the candidate associated with a confirmation code is sufficient to recover voting intent. Of critical importance, therefore, are the trust assumptions surrounding the entities responsible for printing ballots.

In this chapter we tackle the problem of designing a trustworthy optical-scan voting system offering the following desirable properties in full combination:

1. **Distributed trust:** No single party, *including* the ballot printer(s), gains an advantage in deducing how a voter voted or in linking a receipt to its corresponding

clear-text vote. This is a vital requirement of any secret ballot election employing the receipt paradigm.

2. **Single layer ballot form:** A ballot is a single sheet of paper with a *fixed order* candidate list¹ and the voter marks the optical scan ovals *directly beside* their chosen candidate. Multi layer ballots are an artifact of cryptographic voting, requiring voters to re-learn how to cast a ballot. Our experience in running real-world cryptographic elections—both with single layer and with multi layer ballot forms—has indicated to us that multi layer ballots are more cumbersome for voters and more difficult to administer for election officials.
3. **Human-readable paper audit trail:** Pursuant to the legal requirements of many jurisdictions, voting intent remains plainly evident on cast ballot forms. Such an audit trail also allows for recoverability in the event of lost or forgotten cryptographic keys or other unforeseen errors.
4. **Public paper audit trail:** The collection of cast ballot forms (i.e., the *paper audit trail*) can be made public without revealing the link between receipt and clear-text vote. A public audit paper trail may also be a legal requirement and is critical in protecting ballot secrecy during a manual recount.

In this chapter we propose Scantegrity 3-D, a two-party version of Scantegrity II that meets each of the four requirements above. Some of these properties have been examined in the literature, but no proposal has achieved all of them. Scantegrity and Scantegrity II achieve 2 and 3. Prêt à Voter and Scratch & Vote achieve 2 and 4 [CRS05, RS06, AR06, XSH08], two Punchscan variants achieve only 4 [CPSC07, Kub06], and each of Split-Ballot Voting, ClearVote and Kusters *et al.* achieve 1 and 4 [MN07, PC10, KTV09]. A proposal due to Benaloh [Ben08] achieves 2, 3, and 4.

Key Differences between Scantegrity II and Scantegrity 3-D. Scantegrity 3-D structurally differs from Scantegrity and Scantegrity II in two important ways. Firstly, we use a composition of *two* secret master permutations created independently by two non-colluding parties for the audit dataset. Each party is responsible for issuing its own proof on its own master permutation. Roughly speaking this might be viewed as the composition of two independently generated Scantegrity audit datasets.

Secondly, ballot and receipt information is split into two shares such that any one share does not provide sufficient information to break ballot secrecy. In both cases techniques for secure two-party computation are used to generate election data instead of a single trusted

¹There are also potential advantages to using ballots with randomized candidate lists. Our system can accommodate this approach with minor protocol changes.

computer. Physical security mechanisms are used to enforce separation of shares during ballot printing. Protocol changes are mostly internal to the election officials, preserving the voter experience and audit procedures of Scantegrity (aside from having to check twice as many proofs).

Contributions. The contributions of this chapter are summarized as follows:

- **Basic System:** a two-party version of Scantegrity for creating ballot forms with randomized confirmation codes that meets properties 1, 2, and 3. It relies on a private paper audit trail and an in-person physical dispute-resolution procedure.
- **Improved System:** a two-party version of Scantegrity II that uses ‘self-blanking’ invisible ink confirmation codes. It improves on the basic system by allowing the paper audit trail to be made public, thereby achieving all four properties. In addition it offers an *informational* dispute-resolution allowing disputes to be resolved based on knowledge of a confirmation code (as opposed to physical possession of a receipt).

10.2 Preliminaries

10.2.1 Physical Primitives

End-to-end verifiable ballots often employ physical security methods as part of the receipt creation process. The use of physical security mechanisms can be contentious due to inherent questions regarding their cost, feasibility, and real-world security properties. However, there is precedent for protocols built around *ideal* physical security mechanisms (cf. [GKR08, MN05]). We assume that the physical security mechanisms (cf. Section 2.1.6) used in this chapter—namely invisible ink and document authentication—function ideally. Broadly speaking, the ballot secrecy properties of our systems reduce to those of Scantegrity’s when the physical security mechanism fail.

10.2.2 Cryptographic Primitives

A Scalar Homomorphic XOR with Exponential Elgamal. For two bits $m_1, m_2 \in \{0, 1\}$ and their associated encryptions, we describe a method based on exponential Elgamal to implement a scalar homomorphic operation $\tilde{\oplus}$ for which $\llbracket m_1 \rrbracket \tilde{\oplus} m_2$ produces a ciphertext that encrypts the bitwise xor of the associated plaintext bits, i.e., $\llbracket m_1 \oplus m_2 \rrbracket$.

The first party constructs a ciphertext $c = \langle c_1, c_2 \rangle = \langle g^r, g^m y^r \rangle$ for $m \in \{0, 1\}$ and transmits c to the second party. Recalling the ciphertext rerandomization function $\text{ReRand}()$ (cf. 2.1.7), the second party will select their bit $m' \in \{0, 1\}$ and compute the partially-homomorphic xor, $c \tilde{\oplus} m' = \text{PHX}(c, m')$ where

$$\text{PHX}(c, m) = \begin{cases} \text{ReRand}(c) & m = 0 \\ \text{ReRand}(\langle c_1^{-1}, g^1 c_2^{-1} \rangle) & m = 1 \end{cases} .$$

This scheme is essentially the same as the inversion scheme used by Neff in [Nef04]. Importantly, this approach alone only provides security against passive adversaries: the first party could construct a malformed ciphertext, while the second party could throw away the first party's contribution all together. The simplest way to provide integrity would be to run a cut-and-choose protocol whereby both parties output numerous instances of $\langle c_1, c_2 \rangle = \text{PHX}(c, m')$ and conduct a coin-toss protocol to select some instances to challenge. Parties would open the challenged instances (revealing their respective message bits and random factors), and retain the unopened ones for further use.

Commitments. We use a cryptographic commitment scheme (cf. Section 2.1.7) to commit to permutations as part of a cut-and-choose proof of shuffle. The dispute resolution procedure in the improved system requires the prover to either unveil (i.e., *de-commit*) to the code, or alternatively to issue a *non-interactive proof of plaintext inequality*. A commitment inherent to IND-CPA secure encryption fits this dual role. Here a sender *commits* to a message m by posting its encryption $\llbracket m \rrbracket = \text{Enc}(m, r)$. Later the commitment can be unveiled when the sender reveals an m', r' , allowing anyone to verify $\text{Enc}(m', r') = \llbracket m \rrbracket$, and hence $m' = m$. This approach is commonly used in several voting schemes (e.g., [Ben07, AR06, SDW08]).

10.2.3 Participants

There are several entities that participate in the election.

- A set of **voters** with the authority to cast a ballot in the election, optionally construct a privacy-preserving receipt of their vote, and optionally participate in an election audit,
- An **election operations commission** \mathcal{C} with the capability and authority to organize and run an election, operate a polling place, optically scan ballots, report results, act as a custodian of the cast ballot record, and participate in an in-person dispute resolution procedure,

- Two independent **ballot printers** $\mathcal{P}_1, \mathcal{P}_2$ who possess the capability and authority to print documents in the untrusted printing model and participate in a secure (cryptographic) two-party computation,
- An election **scrutineer** \mathcal{S} with the authority to audit the correctness of printed ballots relative to their cryptographic representation. Additionally \mathcal{S} acts as a proxy for voters during disputes with \mathcal{C} to protect their identity. In practice there might be any number of election auditors, representing the candidates or other democracy groups.

As a fundamental requirement of our security model, we assume that neither printer nor election commission collude with one another.

10.3 The Basic System

The basic system produces a public and universally verifiable cryptographic proof attesting to the correctness of the election’s outcome. This correctness proof is based on the cut-and-choose techniques of Scantegrity/Scantegrity II. Without loss of generality we consider a single-contest election involving n ballots² and m candidates. The basic system involves several protocols. The protocols `generateBallots`, `preElectionPrep`, `postElectionPrep` encompass the preparation for the public election audits. Note that each of these protocols taken individually is only secure in an *honest-but-curious* setting. To make them robust against an active adversary we make use of a set of *audit* protocols `proveScan`, `proveReceipt`, `provePrinting` and `resolveDispute`. A summary of notations used is presented in Table 10.1.

n	Number of ballots to print	T	List of all ballot-tuples
m	Number of candidates	BallotTable	Table of ballot information
d	Bit-length of ballot-id	ReceiptTable	Table of receipt information
L	List of candidate names	MP_1/MP_2	Printer 1/2’s master permutation
Σ	Confirmation code alphabet	π/ρ	Random perm’ns composing to MP_1
α	Soundness parameter	σ/τ	Random perm’ns composing to MP_2
b/B	Ballot-id/list of ...	MidMarks	Intermediate mark state list
r/R	Receipt-id/list of ...	MidMarksP1	\mathcal{P}_1 ’s intermediate mark state list
c/C	Confirmation code/list of ...	MidMarksP2	\mathcal{P}_2 ’s intermediate mark state list
μ	Mark-state of opscan oval	eid	Election-unique identifier

Table 10.1: Notations used to describe Scantegrity 3-D.

²The number of ballots printed is the total number of voters times a *heuristically* chosen expansion factor to account for audited and spoiled ballots.

The Ballot. The basic optical-scan paper ballot form has a pre-printed, fixed-order candidate list $L = \{l_1 \dots l_m\}$. Adjacent to each candidate is an optical scan oval with a *mark state* $\mu \in \{0, 1\}$ corresponding respectively to whether the oval was unmarked or marked. The ballot form is separated into two regions by a perforation. The top constitutes the *ballot portion*, and the bottom is the *receipt portion*. An alphabet Σ of m confirmation codes is defined. Each optical scan oval (and hence each candidate) is associated with a confirmation code drawn independently at random, and without replacement, from Σ . A *ballot-id* b is a d -bit³ vector printed on the ballot portion. An independent *receipt-id* r is printed on the receipt portion. The first printer prints the receipt-ids under a scratch-off coating and the second prints the confirmation codes. Both printers will jointly print the ballot-id in invisible ink. Printing of the ballot- and receipt-ids is done such that each printer only knows what *it* prints (and not what its counterpart prints). The basic ballot is depicted in Figure 10.1(a).

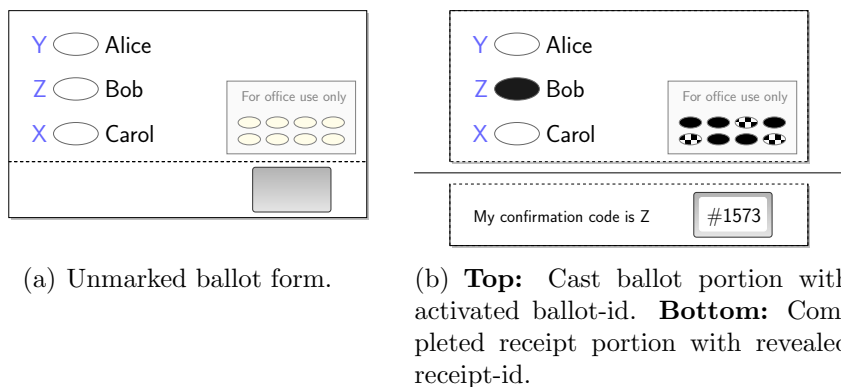


Figure 10.1: **Basic ballot:** Optical-scan ballot form with *ballot portion* (top) and tear-off *receipt portion* (bottom) depicting a randomized confirmation code list, a unique ballot-id printed in *invisible ink visual-crypto* and a unique receipt-id beneath a scratch-off coating. Ballot printing is distributed between two printers such that neither can match receipts with cast ballots.

Ballot Tuple. A ballot is fully specified by the tuple $\{b, r, c\}$, which denotes the association between a unique *ballot-id* bit vector $b \in \{0, 1\}^d$, a unique *receipt-id* $r \in \{1 \dots n\}$, and a random permutation of *confirmation codes* $c = \pi(\Sigma)$ for a permutation π drawn independently and uniformly at random from the set of possible permutations of Σ .

³Since in the basic scheme ballot-ids are the xor of random bit vectors, d is chosen to be large enough so as to make duplicate ballot-ids highly unlikely.

10.3.1 Election Preparation

The election is initialized as follows: election commission \mathcal{C} initializes a *public bulletin board* \mathcal{BB}^4 and a unique election identifier eid . Printers \mathcal{P}_1 and \mathcal{P}_2 jointly run DKG (cf Chapter 2.1.7). They post the public key Y to public bulletin board \mathcal{BB} and retain their respective private key shares x_1, x_2 . This list of public parameters $\text{pubParam} = \{n, m, d, L, \Sigma, \alpha, eid, Y\}$ is posted to \mathcal{BB} . All functions/protocols accept pubParam as input.

Ballot Tuple Creation. The printers now jointly generate encrypted ballot tuples by running `generateBallots`. This protocol is given in Algorithm 10.1.

Ballot Printing. The n ballot forms are printed in three steps. For each ballot-tuple a paper ballot is prepared in the following order:

- **Static background:** directions, candidate names, etc, printed in black ink,
- \mathcal{P}_1 's **share:** the receipt-id is printed and concealed under scratch-off coating, \mathcal{P}_1 's share of the ballot-id is printed in invisible ink visual-crypto,
- \mathcal{P}_2 's **share:** the confirmation codes are printed in regular ink, \mathcal{P}_2 's share of the ballot-id printed in invisible ink visual-crypto over \mathcal{P}_1 's share.

The completed ballot forms are then randomly shuffled and delivered into the custody of the election commission \mathcal{C} . Throughout the ballot printing and voting phases the printers will conduct random audits of ballot forms to ensure their authenticity and to look for signs of tampering (e.g, to catch if someone reveals the secret information then replaces the ballot with a replica). Note that if either printer prints something *other* than their contribution in `generateBallots` (e.g., if a printer prints an all-black VC pixel), this will be caught in `provePrinting` with statistical confidence dependent on the number of audited ballots.

Pre-Election Proof Preparation. The printers initialize the public audit dataset and cut-and-choose correctness proofs by running `preElectionPrep`. This protocol is given in Algorithm 10.2.

Voting and Receipt Creation. An individual wishing to vote shall attend the polling place and authenticate themselves to \mathcal{C} . All qualified and authenticated individuals (i.e., voters) are then eligible to receive a ballot. The voter selects a ballot form at random

⁴Typically modelled as an append-only broadcast channel with state (cf. [BF85]).

Algorithm 10.1: generateBallots

Participants: Printers $\mathcal{P}_1, \mathcal{P}_2$

1 **Printer \mathcal{P}_1 should:**

2 **for** $i \in \{1 \dots n\}$ **do**

3 Encrypt vectors of random bits: $B'(i) \leftarrow (\text{Enc}(\text{randBit}), \dots, \text{Enc}(\text{randBit}))$;

4 Post a non-malleable commitment to each `randBit` along with the random factor used to encrypt it.

5 Encrypt and shuffle receipt-ids: $R \leftarrow \text{Shuffle}(\text{Enc}(1), \dots, \text{Enc}(n))$;

6 Output cryptographic commitments to B' and R ;

7 **Printer \mathcal{P}_2 should:**

8 **for** $i \in \{1 \dots n\}$ **do**

9 Randomly shuffle and encrypt code confirmation codes:

10 $C(i) \leftarrow \text{Shuffle}(\text{Enc}(\Sigma(1)), \dots, \text{Enc}(\Sigma(m)))$;

11 Output a cryptographic commitment to C ;

12 **Both Printers should:**

13 Unveil their respective commitments. Verify the other party's commitments, and terminate the protocol if the verification fails. Otherwise continue to the next step;

14 **Printer \mathcal{P}_2 should:**

15 **for** $i \in \{1 \dots n\}; j \in \{1 \dots d\}$ **do**

16 Homomorphically xor random bits: $b'_1 \dots b'_d \leftarrow B'(i)$;

17 $B(i) \leftarrow (b'_1 \oplus \text{randBit}, \dots, b'_d \oplus \text{randBit})$;

18 Post a non-malleable commitment to each `randBit` along with the random factor used in computing the xor.

19 Output B to \mathcal{P}_1

20 *Remark: $\text{Shuffle}(X)$ applies a permutation to a list X , drawn independently and uniformly randomly from the set of permutations of size $|X|$. `randBit` returns a single bit drawn independently and uniformly at random. It is possible that \mathcal{P}_2 might attempt to maliciously select its bits as a function of \mathcal{P}_1 's. However \mathcal{P}_2 will not know (beyond a guess) what to print on the ballot, and will be caught in `ProvePrinting` with statistical certainty.*

from a stack of unmarked ballot forms and takes it, a regular (black) marking pen, and a privacy sleeve into a private voting booth. The voter marks the oval next to their preferred candidate l_i on the ballot portion. Then, if they so choose, the voter creates a receipt of their vote by noting the code letter c_i and writes it in the appropriate space on the receipt portion. The voter then places the marked ballot form into the privacy sleeve and returns it to the poll worker. The poll worker confirms the receipt-id's scratch-off coating is still intact and the ballot-id has not been activated (rejecting the ballot in such a case), then detaches the receipt portion and places it on a table in view of the voter. The ballot portion is then fed into the optical scanner. If the ballot is rejected the receipt portion is retained by the poll worker. If the ballot portion is successfully cast, the receipt portion is returned to the voter and the voting process is complete. A diagram showing completed

Algorithm 10.2: preElectionPrep

Participants: Printers $\mathcal{P}_1, \mathcal{P}_2$ **Public Input:** Candidate list L **Private Input:** Lists of encrypted ballot-ids B , receipt-ids R , and code shuffles C **1 Both Printers should:***//Expand the n ballot tuples into a table of mn rows (one for every candidate on every ballot):***2 for** $i \in \{0 \dots n-1\}$ **do****3** $c_1 \dots c_m \leftarrow C(i);$ **4** **for** $0 \leq j \leq m-1$ **do****5** $T(1, mi+j) \leftarrow B(i);$ **6** $T(2, mi+j) \leftarrow \text{Enc}(L(j+1));$ **7** $T(3, mi+j) \leftarrow R(i);$ **8** $T(4, mi+j) \leftarrow c_j;$ *// \mathcal{P}_1 followed by \mathcal{P}_2 using master permutations MP1 and MP2 respectively:***9** $T' \leftarrow \text{Mix}(T);$ *//Create ballot and receipt tables:***10** ;**11** BallotTable $\leftarrow \text{DDec}(T'(1 \dots 2, :));$ **12** ReceiptTable $\leftarrow \text{DDec}(\text{Mix}(T'(3 \dots 4, :)));$ **13** Post BallotTable, ReceiptTable to \mathcal{BB} *//Prepare cut-and-choose proof of correspondence between elements in the ballot and receipt tables:***14 Printer \mathcal{P}_1 should:****15 for** $i \in \{1 \dots \alpha\}$ **do****16** Choose $\pi_i \in_R \Pi_{mn};$ **17** Set ρ_i such that $\rho_i \circ \pi_i = \text{MP}_1;$ **18** Post Commit(π_i), Commit(ρ_i) to \mathcal{BB} **19 Printer \mathcal{P}_2 should:****20 for** $i \in \{1 \dots \alpha\}$ **do****21** Choose $\sigma_i \in_R \Pi_{mn};$ **22** Set τ_i such that $\tau_i \circ \sigma_i = \text{MP}_2;$ **23** Post Commit(σ_i), Commit(τ_i) to \mathcal{BB} **24 Remark:** Let $x \in_R \Pi_y$ denote a permutation function x drawn independently and uniformly at random from the set of permutations of list of y elements. Let $\text{MP}_1, \text{MP}_2 \in_R \Pi_{mn}$. Then for $i \in \{1 \dots \alpha\}$, we have $\tau_i \circ \sigma_i \circ \rho_i \circ \pi_i = \text{MP}_2 \circ \text{MP}_1$.

ballot and receipt portions is depicted in Figure 10.1(b).

Timing Attacks. In some jurisdictions, poll workers keep a poll book of voter identities in the *order they voted*. If the scanner were to likewise maintain the order of cast ballots it, taken along with the poll book, would compromise ballot secrecy. Since in our case the ballot is drawn at random from the pile, and the poll worker does not see the ballot- or receipt-ids, this threat can be mitigated by having voters cast ballots into a ballot box at

the polling place and then scanning them later at a central location.

Election Close-down. After the election, \mathcal{C} posts the preliminary election tally obtained by the optical scanners. This procedure along with reporting results, declaring of winners, and any judicially mandated manual recounts are all done entirely in accordance with pre-existing election procedure/law, and are not directly part of this protocol.

Post-Election Proof Preparation. After the election \mathcal{C} populates the **BallotTable** with the mark state information collected by the optical scanners. With this data the printers and can now finalize the cut-and-choose correctness proof by running **postElectionPrep**. This protocol is given in Algorithm 10.3. A diagram showing the relationship between the various mark lists is given in Table 10.2.

Algorithm 10.3: postElectionPrep

Participants: Election Commission \mathcal{C} , Printers $\mathcal{P}_1, \mathcal{P}_2$
Private Input: Secret Master permutations MP_1, MP_2 , Scanned Cast Ballots

//Populate BallotTable with scanner data

1 **Election commission \mathcal{C} should:**

2 **foreach** $\{b, s, \mu\}$ recorded by scanner **do**

3 Find i for which $\text{ballotTable}(1, i) = b$;

4 and $\text{ballotTable}(2, i) = s$;

5 $\text{ballotTable}(3, i) \leftarrow \mu$

6 Post $\text{ballotTable}(3, :)$ to \mathcal{BB} .

//Propagate marks from BallotTable to ReceiptTable

7 **Printer \mathcal{P}_1 should:**

8 $\text{MidMarks} \leftarrow MP_1(\text{BallotTable}(3, :));$

9 Post MidMarks to \mathcal{BB} **for** $i \in \{1 \dots \alpha\}$ **do**

10 $\text{MidMarksP1}_i \leftarrow \pi_i(\text{BallotTable}(3, :));$

11 Post MidMarksP1_i to \mathcal{BB}

12 **Printer \mathcal{P}_2 should:**

13 $\text{ReceiptTable}(3, :) \leftarrow MP_2(\text{MidMarks});$

14 Post $\text{ReceiptTable}(3, :)$ to \mathcal{BB} . **for** $i \in \{1 \dots \alpha\}$ **do**

15 $\text{MidMarksP2}_i \leftarrow \sigma_i(\text{MidMarks});$

16 Post MidMarksP2_i to \mathcal{BB}

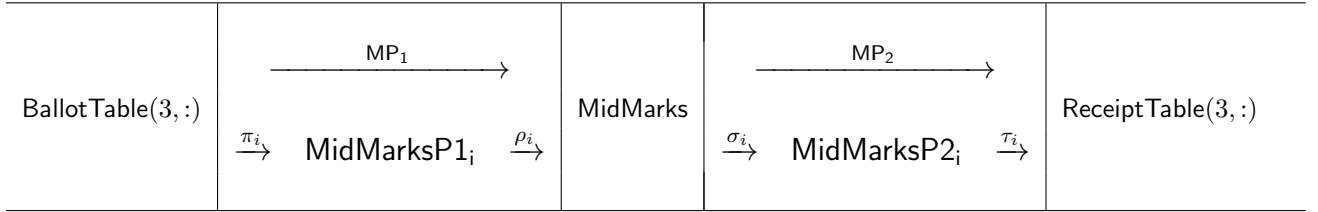


Table 10.2: Diagram of Mark Transformations.

10.4 Verifying the election

There are three simultaneous properties that must be proven in order for the overall results to be proven correct. These audits include:

- **Proving correct mark-state reporting by \mathcal{C} :** Using their receipt, a voter \mathcal{V} checks whether \mathcal{C} correctly registered their vote by running `proveScan`,
- **Proving mark-state propagation by $\mathcal{P}_1, \mathcal{P}_2$:** The printers prove to any interested party that they honestly applied their master permutations to mark state information in `BallotTable` by running `proveReceipt`,
- **Proving printed ballot forms match \mathcal{BB} :** A scrutineer \mathcal{S}^5 runs `provePrinting` with the printers to verify that the ballot tuple information conveyed by the paper ballot forms *matches* the ballot tuple representation in \mathcal{BB} . Audited ballots are *spoiled* and not counted.

The audit protocols `proveScan`, `provePrinting` and `proveReceipt` employed by the basic scheme are below in listed in Algorithms 10.4, 10.5 and 10.6 respectively.

10.4.1 Dispute Resolution

We now describe the dispute resolution procedure for the basic scheme presented in Section 10.3, which is based on the in-person procedure of Scantegrity (cf. Chapter 3), modified to enhance ballot secrecy.

In order to conceal the identity of the voter from \mathcal{C} (who has their ballot) we assume there exists a scrutineer \mathcal{S} that will function as a proxy for the voter during the procedure. Note that the voter will have to trust \mathcal{S} to honestly follow the protocol.

Note however \mathcal{S} does not learn how the voter voted. The dispute resolution procedure for the *basic scheme* is as follows:

⁵A scrutineer is not strictly necessary. Voters themselves may choose to initiate this audit, although in our experience they rarely do!

Algorithm 10.4: proveScan

Participants: Any voter \mathcal{V} who created a receipt

Public Input: ReceiptTable, \mathcal{V} 's receipt $\{r, c_v\}$

// Check receipt against ReceiptTable:

```
1 Voter  $\mathcal{V}$  should:  
2   Find row  $i$  for which ReceiptTable(1,  $i$ ) =  $r$ ;  
3   and ReceiptTable(2,  $i$ ) =  $c_v$ ;  
4   if ReceiptTable(3,  $i$ ) = 1 then  
5     | ACCEPT  
6   else  
7     | Run resolveDispute
```

1. The a scrutineer \mathcal{S} , acting on behalf of a voter, transmits the voter's receipt-id r_v to \mathcal{P}_1 ,
2. \mathcal{P}_1 finds the row in R that contains the encryption r and transmits this index i to \mathcal{P}_2 ,
3. \mathcal{P}_1 and \mathcal{P}_2 each (privately) send to \mathcal{C} the bit vectors they used to construct $B(i)$,
4. \mathcal{C} computes the bitwise xor of the received bit vectors and locates the ballot portion with the resultant ballot-id,
5. \mathcal{S} places their receipt portion in a privacy sleeve that hides the receipt-id,
6. \mathcal{S} and \mathcal{C} continue with the Scantegrity dispute resolution procedure (cf. Chapter 3).

A Possible Dilemma. In many cases the voter will be found to have made a transcription error. However a major dilemma arises from this procedure when it is that case that \mathcal{C} has misreported the code: the voter must give up ballot secrecy to prove to the public that \mathcal{C} is in error.

One of our fundamental requirements for ballot secrecy is that no one be permitted to know the association between ballot-id and receipt-id. If the voter wrote down their code incorrectly, this is not a problem: \mathcal{C} can prove it without needing to know the receipt-id. However if \mathcal{C} reported the code incorrectly, then the receipt-id would need to be made public to prove the discrepancy between the physical and electronic records. This violates ballot secrecy as we have defined it. Still, in the case that the ballot-id/receipt-id association needs to be revealed, it may still suffice if the association between *voter identity* and receipt-id is suppressed. This however would essentially require the voter to never show their receipt to anyone. We leave solving this dilemma to future work, noting that it is

Algorithm 10.5: proveReceipt

Participants: Printers $\mathcal{P}_1, \mathcal{P}_2$ and any interested party

Public Input: A vector Challenge of α challenge bits.

```
1 Both Printers should:
2   for  $i \in \{1 \dots \alpha\}$  do
3     if Challenge( $i$ ) = 0 then
4        $\mathcal{P}_1$  unveils commitment to  $\pi_i$ ;
5        $\mathcal{P}_2$  unveils commitment to  $\sigma_i$ 
6     if Challenge( $i$ ) = 1 then
7        $\mathcal{P}_1$  unveils commitment to  $\rho_i$ ;
8        $\mathcal{P}_2$  unveils commitment to  $\tau_i$ 
9   All decommitment information is posted to  $\mathcal{BB}$ .
10 Anyone can:
11   Run verifyCommit on all of the unveiled commitments;
12   for  $i \in \{1 \dots \alpha\}$  do
13     if Challenge( $i$ ) = 0 then
14       Check:  $\pi_i(\text{BallotTable}(3, :)) = \text{MidMarksP1}_i$ ;
15       Check:  $\sigma_i(\text{MidMarksP2}_i) = \text{ReceiptTable}(3, :)$ 
16     if Challenge( $i$ ) = 1 then
17       Check:  $\rho_i(\text{MidMarksP1}_i) = \text{MidMarks}$ ;
18       Check:  $\tau_i(\text{MidMarks}) = \text{MidMarksP2}_i$ 
19 Remark: Challenge is generated by a public coin toss, or the Fiat-Shamir heuristic when appropriate (e.g., for  $\alpha > 80$ .)
```

mooted by the use of an informational dispute resolution process. We now present an improved system with such an informational dispute resolution procedure.

10.5 Improved System

In this section we present a system that improves upon the basic system in two ways: First, it replaces the physical dispute resolution procedure with an *informational* dispute procedure. Second, the collection of cast ballots (i.e., the paper audit trail) can be viewed publicly without compromising ballot secrecy.

Informational Dispute Resolution. The dispute resolution procedure of the basic system is inefficient and time consuming. Chaum et al. proposed the notion of invisible ink confirmation codes in Scantegrity II as an *informational* means of resolving dispute. Under this approach, codes are printed in invisible ink, and only revealed to the voter if

Algorithm 10.6: provePrinting

Participants: Printers $\mathcal{P}_1, \mathcal{P}_2$, Scrutineer \mathcal{S}

Public Input: A printed ballot chosen at random by \mathcal{S}

1 **Scrutineer \mathcal{S} should:**

- 2 | Activate/scratch-off hidden areas on ballot form to reveal ballot tuple;
- 3 | Post ballot tuple $\{b, L, r, c\}$ to \mathcal{BB}

4 **Printer \mathcal{P}_1 should:**

- 5 | **foreach** BallotTable(1, i) = b **do**
- 6 | | Unveil the commitments to \mathcal{P}_1 's share of b (i.e., ballot id bits and associated random factors);
- 7 | | **foreach** $j \in \{1 \dots \alpha\}$ **do**
- 8 | | | Post $i_{\pi_j} \leftarrow \pi_j(i)$;
- 9 | | | Post $i_{\rho_j} \leftarrow \rho_j(i_{\pi_j})$

10 **Printer \mathcal{P}_2 should:**

- 11 | **foreach** i_{ρ_j} **do**
- 12 | | Unveil the commitments to \mathcal{P}_2 's share of b (i.e., ballot id bits and associated random factors used to compute the xor with \mathcal{P}_1 's share);
- 13 | | **foreach** $j \in \{1 \dots \alpha\}$ **do**
- 14 | | | Post $i_{\sigma_j} \leftarrow \sigma_j(i_{\rho_j})$;
- 15 | | | Post $i_{\tau_j} \leftarrow \tau_j(i_{\sigma_j})$

16 **Anyone can:**

- 17 | Run verifyCommit on all of the unveiled commitments;
 - 18 | Recompute ballot-id b using the unveiled id bits and associated random factors and ensure it matches both the electronic and printed versions;
 - 19 | **foreach** BallotTable(1, i) = b **do**
 - 20 | | **foreach** $j \in \{1 \dots \alpha\}$ **do**
 - 21 | | | Output an error and exit if the following does not hold: BallotTable(3, i) = MidMarksP1 $_j(i_{\pi_j})$ = MidMarks(i_{ρ_j}) = MidMarksP2 $_j(i_{\rho_j})$ = ReceiptTable(3, i_{τ_j});
 - 22 | Output 1;
-

marked. Assuming the code space is sufficiently large so as to make successful random guess unlikely, then knowledge of *any* valid code can be taken as evidence that a voter correctly created their receipt. Any discrepancy found between a receipt and the **ReceiptTable** can then be attributed to \mathcal{C} (assuming the other correctness proofs are valid). In the improved system, we create and print the codes using a *private printing* protocol. Thus the role of invisible ink is twofold: it restricts the voter's knowledge of unmarked codes *and* it prevents the printers from linking receipts to votes.

Public Paper Trail. Invisible ink confirmation codes require a code space that makes random guessing statistically unlikely. For example Scantegrity II proposes a 3-digit code

(making a random guess successful 0.1% of the time times the number of candidates). However in the presence of unique (or semi-unique) codes, access to cast ballots coupled with the public audit dataset is sufficient (or nearly sufficient) to allow *any* observer to link receipts to clear-text votes. This not only means that the paper ballot record must be kept *secret*, but further that the custodian of the ballot record (i.e., \mathcal{C}) is trusted with knowledge of how voters voted. This is one of the major limitations of Scantegrity II. To address this privacy weak-spot, we require a method for not only privately printing a confirmation code, but for displaying it *only while the voter is in the booth*. In the presence of “disappearing” codes, not only can we offer distributed trust with respect to \mathcal{P}_1 , \mathcal{P}_2 and \mathcal{C} , but we can also make the paper ballot record public.

10.5.1 Self-blanking Confirmation Codes

We propose a method for printing of confirmation codes that is self-blanking (i.e., the message is only temporarily visible). The standard invisible ink described by Scantegrity II activates *instantaneously*. That is to say, the chemical reaction responsible for the ink’s pigmentation completes on the order of milliseconds. As previously suggested in Chapter 4, a *slower* reacting ink might be made by the addition of an *anti-catalyst*. This substance, if present, can slow down pigmentation by seconds or minutes (depending on design needs). Combining the technique of visual cryptography with such a ‘slow’ invisible ink, we can construct a self-blanking pixel (see Table 10.3). Finally, combining self-blanking pixels with the two-party oblivious printing protocol of Chapter 9, we can print confirmation codes that are both distributed and self-blanking.













a	b	$VC(a)$	$VC(b)$	Result when activated		
				$t = 0$	$t > 0$	$t \gg 0$
0	0	$\square \emptyset$	$\emptyset \square$			
0	1	$\square \emptyset$	$\square \emptyset$			
1	0	$\emptyset \square$	$\emptyset \square$			
1	1	$\emptyset \square$	$\square \emptyset$			

Table 10.3: **Self-blanking VC Pixel.** Two sub-pixels contain invisible ink. Each party applies an anti-catalyst (white box) to *one* sub-pixel. Sub-pixels containing this substance darken more slowly than those without ($t = 0$ is the moment of activation). Eventually all sub-pixels darken “blanking” the pixel’s value.

10.5.2 The Improved Ballot

The improved ballot differs from the basic ballot in that it makes use of *self-blanking invisible ink* confirmation codes. The codes are printed inside the optical scan ovals in *self-blanking invisible ink*. When the voter marks an oval using the specially provided activator pen, the confirmation code is revealed allowing the voter (finite) opportunity to write down the code on their receipt. Eventually the oval darkens completely indicating *that* the oval was chosen by the voter, but not what the confirmation code was (see Figure 10.2).

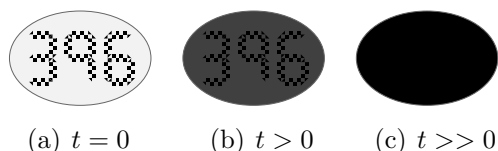


Figure 10.2: **Optical-scan oval with self-blanking confirmation code** after being marked with an activator pen ($t = 0$ is the moment of activation).

10.5.3 Changes to the protocols

The addition of self-blanking invisible-ink confirmation codes induces some changes to the protocols presented in Section 10.3, which are summarized as follows:

- **Ballot tuples:** \mathcal{P}_2 generates ballot-ids. Both printers run a *private printing* protocol to select a confirmation code and distribute it to VC shares,
- **Ballot printing:** \mathcal{P}_2 prints ballot-ids in invisible ink. Both printers print their shares of the confirmation codes using self-blanking visual crypto pixels,
- **Informational dispute resolution:** As in Scantegrity II, the printers only publish the confirmation code corresponding to the voted candidate. In the case of a dispute, the printers jointly issue a non-interactive proof of plaintext inequality between all remaining (unencrypted) codes on the disputed ballot (see Algorithm 10.7).

Changes to generateBallots. The `generateBallots` algorithm of the basic system is adjusted as follows: \mathcal{P}_1 is still responsible for generating and printing a list of unique receipt-ids R . \mathcal{P}_2 now *solely* generates and prints the ballot-ids B . Both printers collaborate to privately generate and print the confirmation codes C using the 2-party oblivious printing protocol of Chapter 9.

Changes to Ballot Printing. As in the basic ballot, \mathcal{P}_1 prints the receipt-ids $R(i)$ and conceals them under scratch-off coating. For each optical-scan bubble, \mathcal{P}_1 applies a solid background of invisible ink and overlays its visual crypto share using the anti-catalyst. Maintaining the ordering, \mathcal{P}_1 transfers the ballots to \mathcal{P}_2 , who prints the ballot-ids $B(i)$ in invisible ink (n.b., without visual crypto). It then applies its VC shares to the corresponding optical-scan bubbles. Printing self-blanking confirmation codes is depicted in Table 10.4.

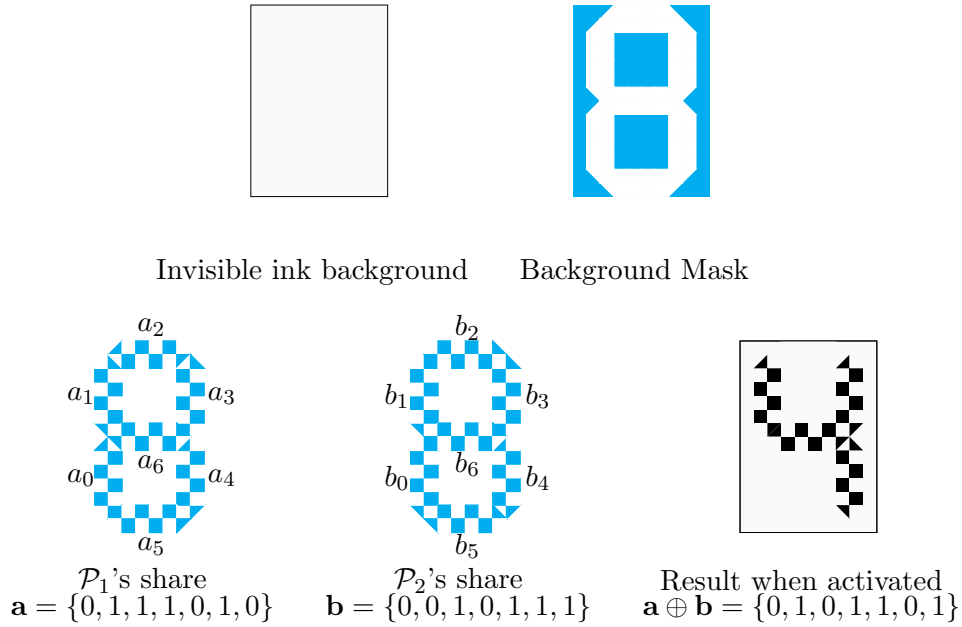


Table 10.4: **Printing a self-blanking invisible ink confirmation code:** \mathcal{P}_1 and \mathcal{P}_2 run a two-party protocol to select a code and distribute it to VC shares (\mathbf{a} , \mathbf{b} respectively). They respectively apply an anti-catalyst over a solid invisible ink background. When activated, the code becomes temporally visible (eventually darkening to all black).

Changes to preElectionPrep. For the most part, the improved system has the same overall structure in terms of the cut-and-choose proof. The `preElectionPrep` is executed in the same way, obviously with the minor difference that the elements of ballot-id list B are now single ciphertexts (as opposed to lists of d encrypted bits), and the elements of receipt-id list R are now encrypted as bit-vectors. When `ReceiptTable` is generated, the receipt-ids are decoded from their 7-segment bit-vector representation into a single integer. However before `ReceiptTable` is posted to \mathcal{BB} , the printers will *encrypt* each of the

confirmation codes (now in integer form) to facilitate the informational dispute resolution procedure. They will save the random factors used in these encryptions for later use.

Changes to proveScan. After the election is complete and the mark state information has been updated, the printers will decrypt and post every code $\text{ReceiptTable}(2, i)$, for which $\text{ReceiptTable}(3, i) = 1$. They will additionally post each associated random factor as proof of decryption.

Changes to provePrinting. When \mathcal{A} audits the printing of a ballot, the codes will only be temporarily visible after activation. This is problematic should \mathcal{A} need to use the ballot as evidence in the event of `provePrinting` fails. There are several ways this might be addressed. One way would be for \mathcal{A} to take a photograph or video of the activated codes however this may not constitute strong enough evidence. Another way would be to require multiple independent auditors to be present when revealing codes. The evidence then would be the testimony of (honest) auditors which may be problematic if a dispute arises over which codes were observed. Finally, we could conjecture the existence of a “fixer” chemical that could halt (or slow) the reaction. This could potentially be accomplished by applying high concentrations of the anti-catalyst substance immediately after activation.

Changes to the Dispute Resolution Procedure. The voter can file a dispute by submitting their receipt-id r and the confirmation code c_v they claimed to have seen. Recall in Scantegrity II, \mathcal{C} responds by unveiling all the commitments to confirmation codes on the dispute ballot. This is acceptable within their security assumptions since \mathcal{C} is trusted to protect ballot secrecy. However since in our system \mathcal{C} is untrusted, the printers must prove in *zero-knowledge* that the disputed code is *not equal* to any of the encrypted codes in `ReceiptTable`, and nothing more. If c_v is a valid confirmation code of receipt-id r , the printers prove the decryption of the associated code in `ReceiptTable`. If c_v is *not* a valid confirmation code, for each row i for which $\text{ReceiptTable}(1, :) = r$, the printers (jointly acting as a single prover) issue a non-interactive *proof of plaintext inequality* between $\text{ReceiptTable}(2, i)$ and c_v . An algorithm for a proof of plaintext inequality, `plaintextInequalityProof`, is given in Algorithm 10.7.

10.6 Security Analysis of the Improved System

To briefly summarize our results, owing to the similarities between systems, we reduce the correctness of the improved system to that of Scantegrity II. Although Scantegrity has been peer reviewed and used in a real election we are not aware of a formal proof of the

Algorithm 10.7: plaintextInequalityProof

Participants: Printers $\mathcal{P}_1, \mathcal{P}_2$ acting as a single prover \mathcal{P} , Voter \mathcal{V}

Public Input: An asserted confirmation code m' , an encrypted confirmation code $c = E(m)$,
public key $\langle g, q, y = g^{sk} \rangle$

1 **Prover \mathcal{P} should:**

 //Encrypt asserted code m'

2 Post $\langle c_1, c_2 \rangle = \langle g^r, my^r \rangle$;

 //Blind quotient of m/m'

3 Select $b \in_r \mathbb{Z}_q^*$ and post $c' = \langle c'_1, c'_2 \rangle = \langle (c_1)^b, (\frac{c_2}{m'})^b \rangle$;

 //Prove knowledge of b

4 Post proof of conjunction on DDH-tuple $\langle c_1, \frac{c_2}{m'}, c'_1, c'_2 \rangle$;

 //Post decryption of c'

5 Post $\langle rb, (\frac{m}{m'})^b \rangle = \langle u, v \rangle$

6 **Verifier \mathcal{V} should:**

7 Verify proof of knowledge of b in step 4;

8 Verify decryption of c' : $\langle g^u, vy^u \rangle = \langle c'_1, c'_2 \rangle$;

9 Verify plaintext inequality of m, m' : $v \neq 1$;

10 Output 0 and exit if any the above do not hold, otherwise output 1.

11 *Remark: This is essentially the plaintext equality test due to Jakobsson and Jules [JJ00] adapted for a single prover. A proof of conjunction of DDH-tuples is due to Chaum and Pedersen [CP92].*

correctness. A formal security proof for the improved system would include a formal security proof for Scantegrity II and is out of the scope for this chapter. A proof of correctness of Eperio offers some insight into how such a proof would proceed (see the Eperio technical report [ECHA12]). With respect to secrecy we present an argument that the improved system protects voter privacy even when one printer is corrupted. Assumptions regarding the physical primitives can be found there as well.

10.6.1 Assumptions

For the security analysis we have to consider the properties of several physical components employed by the improved system of Section 10.5.

Tamper Evidence. Although scratch-off coating and invisible ink function as a form of *physical commitment* scheme, they do not offer the strong assumptions that govern the unveiling of a cryptographic commitment scheme since *anyone* can open such a physical commitment. We make use of *tamper evidence* in the physical commitment setting as a weaker alternative to the hiding property of a cryptographic commitment (cf. [MN05]). Instead of the *hiding* property of cryptographic commitment, a physical commitment ide-

ally has the property that an adversary must actively *tamper* with a document to reveal its secret, which then will be *evident* to the intended recipient.

Scratch-Off Coating. We use scratch-off coating to reversibly conceal some information printed onto a ballot. First we assume that such coating is secure under passive attack, i.e., the message cannot be read without actively tampering with the coating. However anyone can easily remove the coating, so instead of a hiding property we make the *ideal assumption* that revealing the information under the coating can *only* be done in a way that it is *evident*. By checking the integrity of the scratch-off coating anyone can reliably decide whether or not the physical commitment was opened before. If the coating is intact, anyone can be convinced that the content of the commitment is still hidden. In the presence of tamper evidence this type of physical commitment can be viewed as *binding* in the sense that modifying the underlying message would require tampering.

Invisible Ink. For the use of invisible ink we make similar ideal security assumptions as for scratch-off coating with regards to security to passive attack and tamper evidence in the case of active attack. In contrast to scratch-off coating however, it is possible, and actually *desirable*, to be able to add (but not remove) printed information.

‘Slow’ Invisible Ink. The improved system employs invisible ink with delayed activation or *‘slow’ invisible ink*. In addition to our assumptions about normal invisible ink we assume that after a certain time after activation the information printed is no longer readable and thus effectively erased.

10.6.2 Correctness

For the proof of correctness we assume both the voting authority of Scantegrity II, as well as the election commission and both printers of the improved system, are corrupted and under the complete control of an adversary \mathcal{A} . For simplicity we combine the election commission \mathcal{C} and printers \mathcal{P}_1 and \mathcal{P}_2 and denote it as \mathcal{C} . Note that in this case \mathcal{C} knows everything printed in invisible ink or under a scratch-off coating and the physical assumptions only prevent uncorrupted voters from learning information protected that way. We further assume that both voting systems use the same commitment scheme.

Assume there exists an adversary \mathcal{A} able to undetectably cheat in an election run using our improved system. We show how this adversary can then be used to undetectably cheat in a Scantegrity II election by giving a translation from \mathcal{A} to an attack on Scantegrity II. Let $E = \{\mathcal{C}, \mathcal{V}_1 \dots \mathcal{V}_n\}$ be an election system with election committee \mathcal{C} and voters $\mathcal{V}_1 \dots \mathcal{V}_n$.

We say an election system is sound, if, for all adversaries \mathcal{A} , the probability of a verifier accepting an invalid correctness proof is negligible in the security parameter.

Technique. Toward a contradiction, we will show how to use the existence of \mathcal{A} , which implements a correctness attack on our system, to leverage the equivalent attack on the Scantegrity II system. We will accomplish this with rewindable black-box access to \mathcal{A} . At a highlevel, we translate an election being run with Scantegrity II into an election being run with our system. At each phase of the election, we receive output from \mathcal{A} and translate it into the equivalent output in Scantegrity II. We are essentially generating the equivalent election in both systems in parallel. Of course, by using \mathcal{A} , the tally in our system is undetectably incorrect. We show how to translate this into an undetectably incorrect tally in Scantegrity II. Since this should not be possible if Scantegrity II provides correctness, it must be the case that \mathcal{A} cannot exist.

Reduction. We generate the public parameters of the Scantegrity II election, translate them into `PubParam`, and initialize \mathcal{A} with them. This is a direct translation. We then take the output of `preElectionPrep` as generated by \mathcal{A} and attempt to translate it back into the preelection data for Scantegrity II. This translation is possible to do directly but for convenience we will extract, via rewinding \mathcal{A} the permutations π , ρ , σ and τ , which are the permutations between `BallotTable` and `ReceiptTable`. For the extraction to work, we simulate two successful elections to request two different openings of the cut and choose proof for the correctness of the permutation. As it is a 1-out-of-2 proof this is enough to learn the permutations. With this knowledge we prepare the public information for the Scantegrity II election as follows:

Employing the similarities between `BallotTable` in our system and the `S` table of Scantegrity II, we initially group all rows in `BallotTable` with identical ballot-ids, then remove the ballot-id column. This corresponds to the Scantegrity II `S` table. Let this mapping be called M_S . Next we map `ReceiptTable` to the equivalent `Q` table. These two tables are identical except we relabel receipt-ids in `ReceiptTable` as “ballot-ids” in the `Q` table. Let this mapping be M_Q .

Under the assumption that both systems use the same commitment scheme, we directly transfer all commitments as-is. From the permutations extracted from \mathcal{A} and the mappings M_Q and M_S we compute the mapping that maps each cell in table `Q` to one in table `S`.

We do this by composing permutations ρ, τ and preparing `Q`-pointers that correspond to this resultant permutation. Similarly we generate the `S`-pointers from the composition of permutations π, σ . Then we publish the tables `Q`, `R` and `S`.

During the election phase \mathcal{A} has access to ballot choices made by each voter and returns a confirmation code together with the receipt-id to the voter.

After the election phase we query \mathcal{A} for the mark positions in `BallotTable` and `ReceiptTable`. Using M_Q and M_S we translate these positions to the corresponding entries in \mathbf{Q} and \mathbf{S} and publish them. We also publish the incorrect tally given by \mathcal{A} . The audit challenges are directly translated from Scantegrity II into our system. \mathcal{A} responds to the challenges by unveiling the corresponding sub-permutations, either $\{\rho, \tau\}$ or $\{\pi, \sigma\}$, which will be accepted by definition of \mathcal{A} . We then translate this accepting proof into Scantegrity II by unveiling the corresponding \mathbf{Q} - or \mathbf{S} -pointers. At this point a verifier in Scantegrity II will accept the incorrect tally. This fact contradicts the correctness of Scantegrity II—therefore we conclude \mathcal{A} does not exist.

10.6.3 Voter Privacy (Sketch)

For space considerations we only sketch the properties of voter privacy. One aspect of our improved system is that voter privacy is still guaranteed if one printer is corrupted. We claim that the additional information an adversary \mathcal{A} gains by corrupting one printer is insufficient to learn anything about the choice of a single voter.

\mathcal{A} corrupts \mathcal{P}_1 . When \mathcal{A} corrupts \mathcal{P}_1 the additional information gained is the secret key x_1 , a share of each code and for each $i \in \{1 \dots \alpha\}$ the two permutations π_i and ρ_i . Also \mathcal{A} learns all corresponding receipt-ids. The security properties of the encryption scheme prevent \mathcal{A} from decrypting any ciphertexts by only knowing x_1 . During the postelection proof only one of σ_i or τ_i is ever revealed, therefore \mathcal{A} does not learn the master permutation of \mathcal{P}_2 . Thus no information is revealed about the permutation between `BallotTable` and `ReceiptTable` given that the commitment scheme is hiding. Because \mathcal{P}_1 prints its share of ballot tuples first, it does not learn anything about \mathcal{P}_2 's share, which in turn ensures \mathcal{A} learns nothing about the association between `BallotTable` and `ReceiptTable`.

\mathcal{A} corrupts \mathcal{P}_2 . When \mathcal{A} corrupts \mathcal{P}_2 the additional information gained is the secret key x_2 , a share of each code and for each $i \in \{1 \dots \alpha\}$ the two permutations σ_i and τ_i . For the same reasons as above this does not give \mathcal{A} an advantage in breaking voter privacy. Because \mathcal{P}_1 printed the receipt-id and covered it in scratch-off coating, and printed its share of the confirmation codes in invisible ink, \mathcal{A} learns nothing about the association between receipt-ids, codes, and ballot-ids assuming the security properties of these physical primitives as stated in previous sections.

If \mathcal{P}_2 is able to read the receipt-ids (by breaking the security assumption about the scratch-off coating), or the shares of \mathcal{P}_1 (by breaking the security assumptions about invisible ink), or is even able to replace the ballots printed by \mathcal{P}_1 without leaving evidence,

\mathcal{P}_2 learns enough to break voter privacy. In this case the privacy of the improved system reduces to that of Scantegrity II with a corrupted printer.

\mathcal{A} coerces \mathcal{V} . When \mathcal{A} coerces \mathcal{V} , we seek assurance that \mathcal{V} cannot prove how he/she voted to \mathcal{A} . This property is known as coercion resistance. It has been shown by Küsters et al. that Scantegrity II achieves coercion resistance [KTV10]. We do not attempt to prove coercion resistance for our system but given the demonstrated similarities between both the cryptography and the ballot, we would expect a proof of coercion resistance would be easy to construct following their result.

We have to assume that \mathcal{A} does not have unlimited access to the public paper trail. Specifically \mathcal{A} must not be able to recognize any ballot, for example by making an in-depth analysis of the fibre structure as used for ensuring document authenticity. If \mathcal{A} is able to identify a ballot and has made a similar fibre analysis while the ballots were in the custody of a corrupted printer \mathcal{A} would be able to pair a receipt of a coerced voter with the retained part of the ballot in the public paper trail. This problem is not specific to our improved system but always occurs when \mathcal{A} gains enough information to link a paper receipt to a plaintext ballot in the public paper trail. A simple countermeasure would be to restrict the access to the public paper trail.

10.7 Discussion

10.7.1 Technical Challenges

The formulation of invisible inks has many areas for improvement. Although we have made progress in the manufacture of invisible inks in the context of Scantegrity, we observed our ink chemistry led to rapid degradation in the print-heads causing printers to eventually fail after printing only a fraction of a single precinct's worth of ballots. It also seems possible that codes could be passively attacked (i.e., read without activation) under laboratory-based forensic analysis. It would be important to the credibility of invisible ink to have a sense of how costly this would be.

Proper alignment (i.e., registration) of shares has long been a limitation of visual cryptography. Assuming an optical scan oval width of 1 cm and the visual crypto pattern depicted in Figure 10.2, the sub pixels would be on the order of 3 mm wide. Assuming a quality tolerance of $>90\%$ overlap of subpixels between shares, then the printing alignment would require a tolerance on the order of about 0.3mm (in both horizontal and vertical axes). It seems plausible current consumer printers could achieve this. Printing confirmation codes in the improved scheme would require finer granularity. A rough estimate

based on the codes in used in the 2008 Takoma Park election suggests one VC sub-pixel per millimeter might suffice. This would require a precision on the order of $100\mu\text{m}$ which likely exceeds the capability of consumer printing technology. We envision a device that could *simultaneously* scan positional markers on the document and align the print head relative to them in real time. The author is not aware of any existing implementation of this, but note it may be an interesting avenue for future work.

10.7.2 Usability Questions

Sherman et al. [SCC⁺10] have studied some basic usability questions about invisible ink confirmation codes. There are however several additional and potentially important usability questions that would need to be answered before *self-blanking* invisible ink confirmation codes could be used in a real election.

Perhaps the most important question would be to understand how disappearing ink might interfere with the voter marking the ballot as intended. It is certainly possible that the voter's mental model of marking a ballot may be affected by the delayed darkening of the oval. Would this delay represent confusing feedback for the voter while they attempt to confirm whether they successfully *marked the ballot as they had intended to*?

Another important question pertains to the potential pitfall to privacy if the voter leaves the booth too early. The voter would need to be instructed to stay in the booth until the oval has darkened fully (and hence the code has disappeared). The longer it takes for an oval to darken, the more likely it would be that a voter might choose to disregard the instruction and leave the booth anyway. A naïve technical solution would be to dilute the inhibiting substance to speed up the reaction. However this comes at cost of giving the voter a smaller window of opportunity to write down the code on their receipt. This might especially be problematic for multi contest ballots if the voter decides to complete marking the ballot first and *then* record their codes later, as the codes may have disappeared by then.

10.8 Concluding Remarks

In this chapter we presented two systems for verifiable optical-scan voting with single layer ballots and without trusted components. The basic system based on randomized confirmation codes utilizes existing techniques for invisible ink printing. The improved system proposes a novel self-blanking invisible ink, allowing us to construct a system with more efficient dispute resolution procedure and public paper audit trail.

Both systems are two-party protocols. Ultimately however it would be desirable to be able to distribute trust among arbitrarily many printers. With some modification the improved system presented in Section 10.5 could likely be extended to a secure multi-party protocol.

The primary challenge is to develop an effective approach to distributing the ballot printing among more than two printers. This requires a fundamentally different approach from the two-party private printing scheme of Chapter 9, and is the subject of the following chapter.

Chapter 11

Multi-party Oblivious Printing

The blind envy the one-eyed.

Juvenal

This chapter is adapted from published work supervised by Urs Hengartner [ECHA09].

11.1 Introductory Remarks

The previous two chapters outline our research results in designing a trustworthy optical-scan voting system with fully distributed trust. The approach taken in Chapter 9, however, is only offers distribution of trust between two parties only, and a multi-party extension is not forthcoming under that approach. Furthermore, the approach is limited by the fact that the protocol cannot output an associated ciphertext along with the printed document, making the resulting voting protocol presented in Chapter 10 more complex than strictly necessary.

In this chapter we extend oblivious printing to a fully multi-party setting, and tackle the issue of outputting the associated ciphertext. The result allows us to draw a conceptual and procedural circle around the ballot generation and printing phase, which, as we will later show in Chapter 12, allows for a simpler description of the overall protocol, in addition to offering distribution of trust which is comparable to a fully electronic voting scheme.

Contributions. The contributions of this chapter include multi-party protocols, secure against a malicious adversary for:

- Obviously printing the contents of an encrypted plaintext,
- A contrast improvement when generating and obviously printing a randomized message,
- Generating and obviously printing an Elgamal/DSA keypair.

11.1.1 The Oblivious Printing Model

Oblivious printing is a protocol in which a group of printers cooperate to print a secret message. This message can be revealed and read by the intended recipient, but remains unknown to the printers. Oblivious printing is accomplished through a combination of cryptographic and document security techniques. The high level procedure is sketched as follows:

1. **MESSAGE SELECTION:** Printers execute a secure multi-party protocol to select a message (under encryption) from an alphabet of valid messages.
2. **GRAPHICAL SECRET SHARING:** Printers convert the message (under encryption) into a graphical image. Using a dealerless protocol they secret share the pixels between themselves.
3. **INVISIBLE INK OVERPRINTING:** Pixel shares are converted into a visual crypto pattern. Using invisible ink each printer successively prints their share on the same sheet of paper and in a known location/orientation.
4. **MESSAGE RECOVERY:** The recipient of the completed document activates the invisible ink of the combined shares (e.g., using a special activation pen), thereby revealing the message. This step is depicted in Figure 9.2.

We presented a preliminary two-party protocol for oblivious printing of randomized messages based on oblivious transfers in Chapter 9. The techniques presented in this chapter generalize the model to a fully multi-party setting. Additionally this approach allows for the secret message to be simultaneously output as an obviously printed document and as an associated ciphertext allowing greater possibilities for integration into broader protocols.

11.2 Visual Cryptography in this Setting

Recalling the discussion of visual cryptography from Section 2.1.7, we first review the notion of contrast for an n -out-of- n visual crypto scheme. Shamir and Naor [NS94] prove the optimal number of sub-pixels for an (n, n) -VCS is 2^{n-1} . In this scenario, if the dealer wishes to share a black pixel, shares are constructed such that when an authorized set of printers combine their shares, each of the resulting 2^{n-1} sub-pixels will be black. Similarly if the dealer wishes to share a white pixel, one of the resulting sub-pixels will be white (the other $2^{n-1} - 1$ will be black). This is used to define a measure of contrast, α , as being the relative difference in intensity between the combined shares resulting from a white pixel and a black pixel in the original image. The optimal contrast for an (n, n) -VCS is thus $\alpha = \frac{1}{2^{n-1}}$.

11.2.1 Fixed Pattern Visual Cryptography

We make use of some aspects of visual cryptography for the purposes of oblivious printing; however there are several important differences with how it is typically presented in the literature:

1. **INVISIBLE INK SHARES:** Printers successively overprint their shares in invisible ink on a **single sheet of paper**. Activation of the combined invisible ink shares recovers the message.
2. **DEALERLESS SHARE CREATION:** The message is distributed to shares by a multi-party computation.
3. **FIXED SUB-PIXEL PATTERNS:** Each printer has a fixed pair of sub-pixel patterns. Which of the two patterns the printer ends up printing is secret, but the patterns themselves are a public input to the protocol.

We will make use of a set of sub-pixel patterns that implement an XOR operation. Work has been done into visual cryptography in a variety of physically XOR-ing media including interferometers [LNS⁺02], light polarization [THvLT05], and even image reversal using modern photocopy machines [VK04]. In our approach, however, the XOR is approximated in an underlying physical medium (i.e., over-printed shares of invisible ink) that implements an OR.

Definition 11.1. *An n -input visual XOR, n -VCX, describes a set of sub-pixel patterns that visually implement an XOR of the input bits in a physically OR-ing medium.*

Let S be an $n \times 2^{n-1}$ binary matrix for which each column is unique and has an even Hamming weight. Let \overline{S} be the element-wise complement of S . Let sub-pixel pattern matrix Φ be as follows: $\Phi(\ell, 0) = S(\ell, :)$ and $\Phi(\ell, 1) = \overline{S(\ell, :)}$.

For a set of Boolean values $a_1 \dots a_n \in \{0, 1\}$ and their associated logical exclusive-or $a' = \bigoplus_{i=1}^n a_i$, we say the sub-pixel pattern matrix Φ implements an n -input visual crypto exclusive-or, if the sub-pixel pattern produced by overlaying shares $\Phi(1, a_1) \dots \Phi(n, a_n)$ has the following outcome: the total number of black sub-pixels is $2^{n-1} - 1$ if $a' = 0$, and respectively 2^{n-1} when $a' = 1$. If the a_i 's contain an even number of ones (i.e., the XOR is zero), then exactly one of the columns will end up with all 0's (i.e., a white sub-pixel) due to the way the matrix was designed and the pixel will be visually interpreted as white. If the a_i 's contain an odd number of ones (i.e., their XOR is one), all columns will contain a non-zero number of 1's due to the way the matrix was designed and the pixel will be visually interpreted as black. Φ implements an n -VCX.

Example 11.1. A 4-VCX: Let inputs $[a_1, a_2, a_3, a_4] = [1, 0, 0, 1]$ and,

$$S = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

We have $\Phi(1, 1) = [1, 1, 1, 0, 1, 0, 0, 0]$, $\Phi(2, 0) = [0, 0, 1, 0, 1, 0, 1, 1]$, $\Phi(3, 0) = [0, 1, 0, 0, 1, 1, 0, 1]$, and $\Phi(4, 1) = [1, 0, 0, 0, 1, 1, 1, 0]$. When the vectors are OR-ed, it produces the sub-pixel pattern $[1, 1, 1, 1, 0, 1, 1, 1]$. Such a pattern is visually interpreted as intended, i.e., a white pixel with contrast $\alpha = \frac{1}{8}$.

11.3 Obviously Printing an Encrypted Message

In this section we present a protocol for obviously printing the contents of a ciphertext for which the associated plaintext is within a known, bounded, alphabet of m possible valid messages. Given an encrypted plaintext $\llbracket p \rrbracket$, a set of n printers $\mathcal{P}_1 \dots \mathcal{P}_n$, each with a share of the decryption key, will jointly print p as a $(u \times v)$ -pixel image I_p depicting p in a human- or machine-readable form such that no printer learns p . We leave the origin of $\llbracket p \rrbracket$ generic although we envision it as being the output of some other (previous) multi-party computation.

11.3.1 Translation Table

A translation table is defined in which each element is a valid possible message that can be printed and for which each message consists of an association between a plaintext value

and a bitmap that depicts it. The translation table is taken as input to the protocol and is used to facilitate the translation of a message—under encryption—from its plaintext form to its bitmap depiction.

Let translation table T consist of m message pairs representing the set of valid messages that can be printed. Each message pair $\langle t, I_t \rangle \in T$ consists of a plaintext value t in the plaintext domain, and a $(u \times v)$ -pixel monochrome bitmap I_t depicting t as a human- or machine-readable image. Each value $I_t(i, j) \in \{0, 1\}$ corresponds respectively to a white or black pixel.¹ We use the notation $\llbracket T \rrbracket$ to denote the element-wise encryption of T . Each message pair $\langle t, I_t \rangle \in T$, can be regarded as a vector of $uv + 1$ elements, $[t, I_t(0, 0), \dots, I_t(u-1, v-1)]$ where each element is encrypted separately. The initial encryption of each element is taken with a known random factor (e.g., 0). Mixing $\llbracket T \rrbracket$ involves re-randomizing each element and shuffling by the message pair vectors.

In order to facilitate mixing and searching for elements in $\llbracket T \rrbracket$ under encryption, $|T|$ in practice will be small relative to the plaintext domain. Because I_t will be encrypted at the pixel level we note that for practical purposes the image size should be kept small. Using the upsampling technique proposed in Chapter 9, a single alphanumeric character (or digit) can be fully described in 16 (respectively 7) encryptions regardless of the resolution of the visual crypto sub-pixel pattern used.

11.3.2 Setup

Let $\langle \text{DKG}, \text{Enc}, \text{DDec} \rangle$ be an encryption scheme with distributed decryption. Distributed key generation $\text{DKG}(n)$ generates a public key, e , and a private key share, d_i associated with printer \mathcal{P}_i . Encryption $\llbracket m \rrbracket = \text{Enc}_e(m, r)$ is semantically secure and homomorphic in one operation. Distributed decryption $\text{DDec}_{d_i}(c)$ of a ciphertext c is possible with all n printers applying their shares d_i . Without loss of generality we use Elgamal [Ped91].

11.3.3 The Protocol

The protocol for obviously printing a $p \in T$ given $\llbracket p \rrbracket$ is described in Protocol 11.3 and consists of Sub-protocols 11.3(a) and 11.3(b). Briefly, Sub-protocol 11.3(a) encrypts and mixes T and searches it (under encryption) for the entry corresponding to p , outputting the associated encrypted bitmap. The process of searching the encrypted translation table for a value and outputting the associated encrypted image as described in Step 2 of Sub-protocol 11.3(a) is closely related to the Mix and Match system [JJ00]. In Step 1 of

¹Printing uses a *subtractive* color model and thus the plaintext values assigned to color intensities are the reverse of that found in the computer graphics literature.

PROTOCOL 11.3 (Obviously Print p given $\llbracket p \rrbracket$).

Input: Translation table T , encrypted plaintext $\llbracket p \rrbracket$, sub-pixel matrix Φ implementing an $(n+1)$ -VCX, soundness parameter ρ .

Output: A document with a $(u \times v)$ -pixel image depicting p , printed in invisible ink and with contrast $\alpha = \frac{1}{2^n}$.

The protocol:

1. TRANSLATE ENCRYPTED PLAINTEXT INTO ASSOCIATED PIXEL-WISE ENCRYPTED IMAGE: Run Sub-protocol 11.3(a).
2. OBLIVIOUSLY PRINT ENCRYPTED IMAGE: Run Sub-protocol 11.3(b).

Sub-protocol 11.3(b) the printers secret share a pixel by homomorphically XOR-ing it with random bits in a manner similar to the technique used by Cramer et al. [CD01].

SUB-PROTOCOL 11.3(a) (Translate $\llbracket p \rrbracket$ into $\llbracket I_p \rrbracket$).

Input: Translation table T , encrypted plaintext $\llbracket p \rrbracket$.

Output: A $(u \times v)$ pixel-wise encrypted image of p (i.e., $\llbracket I_p \rrbracket$).

The protocol:

1. ENCRYPT AND VERIFIABLY MIX TRANSLATION TABLE T : Each printer participates in a verifiable mix network, which encrypts and shuffles the message pairs $\langle t_i, I_{t_i} \rangle \in T$. The result is denoted $\llbracket T' \rrbracket$.
2. FIND $\llbracket p \rrbracket$ IN $\llbracket T' \rrbracket$: printers search $\llbracket T' \rrbracket$ attempting to locate a $\llbracket t_i \rrbracket$ for which $t_i = p$:
 - (a) For each message pair $\langle \llbracket t_i \rrbracket, \llbracket I_{t_i} \rrbracket \rangle \in \llbracket T' \rrbracket$, the printers perform a test of plaintext-equality between $\llbracket p \rrbracket$ and $\llbracket t_i \rrbracket$.
 - (b) If a match is found, output the corresponding pixel-wise encrypted bitmap $\llbracket I_{t_i} \rrbracket$. If no match is found the protocol terminates and an error message is output.

Remark: Various protocols exist for verifiable mix networks. One efficient and statistically sound technique for multi-column mixing is due to Sako and Kilian[SK95]. The plaintext equality test (PET) is due to Juels and Jakobsson [JJ00].

Finalization Layer. Given n printers note that Protocol 11.3 uses an $(n+1)$ -VCX. An additional “finalization” layer allows the printers to verify the correctness of printing without ever revealing the message. For each pixel, each printer will generate a random bit, and using the sub-pixel pattern matrix, print it in invisible ink. A cut-and-choose proof is performed in Step 2 to demonstrate the printers correctly printed their random bits. Then

an $(n+1)$ -th finalization layer is computed by homomorphically XOR-ing the message bit with each of the random bits. Since the finalization layer is essentially a one-time pad, it can be decrypted without revealing the message. Finally, the finalization layer is printed using black ink, the correctness of which can be verified visually by inspection.

11.3.4 Obviously Printing an Arbitrary Plaintext

In Protocol 11.3 we showed how to obviously print a plaintext $p \in T$ given its encryption. As was previously mentioned, in order to make mixing and searching $\llbracket T \rrbracket$ feasible, $|T|$ will typically be quite small relative to the plaintext space.

We briefly sketch how a solution accommodating an arbitrary message from the plaintext space might be approached. To print an arbitrary p , first the printers would define an alphabet Σ (e.g., the Latin alphabet) for which p could be represented as a string Σ^ℓ . The printers would execute a multi-party pre-protocol to convert $\llbracket p \rrbracket$ into a collection of ciphertexts $\llbracket p_1 \rrbracket \dots \llbracket p_\ell \rrbracket$ for which $p = p_1 \parallel \dots \parallel p_\ell$ (a homomorphic/universally verifiable multi-party protocol for extracting bit-fields under encryption is left to future work). The printers would then run Protocol 11.3 for each p_i , printing the result on the same sheet of paper.

11.4 Obviously Printing a Randomized Message

In this section we present a contrast optimization in the special case where the printers are printing a randomized message $r \in_R T$. Although Protocol 11.3 can also be used for this purpose the protocol presented in this section has a contrast of $\alpha = \frac{1}{2^{n-1}}$ (as opposed to $\alpha = \frac{1}{2^n}$). Protocol 11.3 allows the printers to engage in a cut-and-choose proof of correct printing without revealing p directly. This is done at the expense of contrast: the use of the finalization layer introduces an additional layer forcing the n printers use an $(n+1)$ -VCX, which has half the contrast relative to an n -VCX.

If the message is randomized, then revealing it as part of a cut-and-choose process does not reveal information about the remaining (unactivated) messages. So instead of partially printing ρ copies of a single message p , auditing $\rho - 1$ copies and finalizing the remaining copy, the printers instead obviously print ρ complete and independently random messages, of which they audit $\rho - 1$. The protocol is described in Protocol 11.4.

Arbitrary-length random messages can be built by repeated (independent) executions of Protocol 11.4 on the same sheet of paper, which may be useful in the creation of strong passwords, cryptographic keys or random tokens. Note in this setting the bit-field extraction step outlined in Section 11.3.4 would be unnecessary.

SUB-PROTOCOL 11.3(b) (Obviously Print $\llbracket I_t \rrbracket$).

Input: A $(u \times v)$ pixel-wise encrypted image $\llbracket I_t \rrbracket$, sub-pixel matrix Φ implementing an $(n+1)$ -VCX, soundness parameter ρ .

Output: A document with I_t printed in invisible ink with contrast $\alpha = \frac{1}{2^n}$.

The protocol:

1. OBLIVIOUSLY PRINT ρ INSTANCES OF I_t : For each $1 \leq i \leq \rho$:
 - (a) PRINT A NEW INSTANCE (SHEET): For each pixel $\llbracket I_t(j, k) \rrbracket$:
 - i. POST COMMITMENTS TO RANDOM BITS: Each printer $\mathcal{P}_{\ell \leq n}$ draws a random bit $b_{i,j,k,\ell} \in_R \{0, 1\}$ and broadcasts a non-malleable commitment to it.
 - ii. SECRET SHARE PIXEL: The n printers jointly compute an encrypted finalization pixel $\llbracket f_{i,j,k} \rrbracket = \llbracket t(j, k) \rrbracket \oplus b_{i,j,k,1} \oplus \dots \oplus b_{i,j,k,n}$ using a scalar homomorphic XOR.
 - iii. PRINT SUB-PIXEL PATTERN IN INVISIBLE INK: Each printer $\mathcal{P}_{\ell \leq n}$ records the unique physical characteristics of the paper sheet and overprints the sub-pixel pattern $\Phi(\ell, b_{i,j,k,\ell})$ in invisible ink on the i -th document instance at the position associated with pixel (j, k) .
2. PERFORM CUT-AND-CHOOSE PROOF OF CORRECT PRINTING: The printers select $\rho - 1$ documents at random to audit (see remark). For each chosen sheet:
 - (a) UNVEIL COMMITMENTS: Each printer unveils their uv commitments from Step 1(a)-i.
 - (b) PROVE XOR: Each printer broadcasts their random factor used in computing the scalar homomorphic XOR in Step 1(a)-ii.
 - (c) ACTIVATE INVISIBLE INK: The printers collectively activate the invisible ink revealing the result of Step 1(a)-iii.
 - (d) VERIFY: Each printer performs the following steps. If any of them do not hold, the protocol is terminated and an error message output:
 - i. CHECK COMMITMENTS: Verify commitments produced in Step 2(a).
 - ii. CHECK XOR: Recompute the homomorphic XOR using $\llbracket t(j, k) \rrbracket$ and random factors revealed in Step 2(b) and confirm result equals finalization pixel from Step 1(a)-ii.
 - iii. CHECK PRINTING: For each pixel ensure the combined VC sub-pixel pattern created by the bits revealed in 2(a) corresponds to the printed version revealed in Step 2(c).
 - iv. CHECK PAPER: Authenticate the sheet against those in Step 1(a)-iii.
3. FINALIZE THE REMAINING SHEET:
 - (a) DECRYPT FINALIZATION LAYER: The printers decrypt the finalization pixels $\llbracket f_{i,j,k} \rrbracket$.
 - (b) PRINT FINALIZATION LAYER: The printers authenticate the sheet. If the sheet is not recognized, the protocol terminates and an error message is output. Without loss of generality \mathcal{P}_1 prints the finalization layer: each pixel $\Phi(n+1, f_{i,j,k})$ is printed in black ink at the associated position. The other printers check the finalization layer is printed correctly. The resulting document is securely delivered to its intended recipient.

Remark: See Section 10.2.2 for a description of the scalar homomorphic XOR using exponential Elgamal. The heuristic due to Fiat and Shamir [FS86] can be used to fairly select documents to audit.

PROTOCOL 11.4 (Obliviously Print a Random $r \in_r T$).

Input: Translation table T , sub-pixel matrix Φ implementing an n -VCX, soundness parameter ρ

Output: A document with a $(u \times v)$ -pixel image depicting a random $r \in_r T$, printed in invisible ink and with contrast $\alpha = \frac{1}{2^{n-1}}$. Encrypted plaintext $\llbracket r \rrbracket$.

The protocol:

1. OBLIVIOUSLY PRINT ρ INDEPENDENT RAND. MSGS.: For each $1 \leq i \leq \rho$:
 - (a) SELECT RANDOM MESSAGE PAIR FROM T : Run Step 1) from Sub-protocol 11.3(a) to generate $\llbracket T'_i \rrbracket$. Without loss of generality the printers select encrypted message pair $\llbracket T'_i(0) \rrbracket = \langle \llbracket r_i \rrbracket, \llbracket I_{r_i} \rrbracket \rangle$.
 - (b) OBLIVIOUSLY PRINT $\llbracket I_{r_i} \rrbracket$: Run Step 1a) from Sub-protocol 11.3(b) with the following modifications:
 - Without loss of generality, the first $(n-1)$ printers $\mathcal{P}_{\ell < n-1}$ partially decrypt the secret shared pixel $\llbracket f_{i,j,k} \rrbracket$ created in Step 1(a)-ii by applying their respective shares of the private key.
 - Similar to Step 1(a)-iii each printer $\mathcal{P}_{\ell < n-1}$ overprints their VC sub-pixel pattern $\Phi(\ell, b_{i,j,k,\ell})$. Printer \mathcal{P}_n decrypts the partial decryption of $\llbracket f_{i,j,k} \rrbracket$ and prints $\Phi(n, (b_{i,j,k,n} \oplus f_{i,j,k}))$ in invisible ink.
2. PERFORM CUT-AND-CHOOSE PROOF OF CORRECT PRINTING: The printers select and audit $\rho-1$ documents as in Step 2 of Sub-protocol 11.3(a).
3. OUTPUT REMAINING SHEET: The remaining document $I_{r'}$ is securely delivered to its intended recipient. The associated ciphertext $\llbracket r' \rrbracket$ is output.

11.4.1 Generating and Obliviously Printing a DSA/Elgamal Key-pair

One interesting variation of Protocol 11.4 might be generating and obliviously printing an DSA/Elgamal keypair for which the printers do not know the private key. This could potentially be an interesting approach to building a PKI in which a group of printers acting as a distributed CA issues keypairs in physical form.

The keypair can be rendered in a convenient encoding such as alphanumeric (e.g., Base64) or 2-D barcode (e.g., a QR-code). We note that 2-D barcodes often contain additional error correction information. Creating a valid error-correction code under encryption is something we leave to future work. We present a protocol for generating and obliviously printing a DSA/Elgamal keypair in Protocol 11.5.

PROTOCOL 11.5 (Generate and Obliviously Print a DSA/Elgamal Keypair).

Input: A large prime $p = 2\alpha q + 1$ (for an integer α), a generator $g \in \mathbb{G}_q$, an encoding alphabet Σ (e.g., Base64) for which $|\Sigma|$ is a power of 2.

Output: A document with public key $y = g^{sk}$ printed in black ink, and secret key sk printed in invisible ink.

The protocol:

1. For $0 \leq i < \lfloor \frac{\log_2(q)}{|\Sigma|} \rfloor$:
 - (a) INITIALIZE T_i : For $0 \leq j < |\Sigma|$: Add message pair $\langle g^{j+|\Sigma|i}, I_{\Sigma(j)} \rangle$ to T_i .
 - (b) OBLIVIOUSLY PRINT PRIVATE KEY SEGMENT: Printing on the same sheet each time so as to build a string of characters, run Protocol 11.4 with T_i as input, receiving an (encrypted) segment of the private key $c_i = \llbracket g^{sk_i} \rrbracket$.
2. RECOVER PUBLIC KEY: Printers decrypt $\llbracket y \rrbracket = \llbracket g^{\sum_i r_i} \rrbracket = \prod_i c_i$. Without loss of generality \mathcal{P}_1 prints the result in black ink and other printers confirm the value is correctly printed. The result is securely delivered to the intended recipient.

Remark: If the secret key's bit length does not evenly divide the encoding alphabet, the above loop is run one final time with a reduced alphabet $\Sigma' \subset \Sigma$ where $|\Sigma'| = \lceil \log_2(q) \rceil \bmod |\Sigma|$.

11.5 Mitigating Contrast Drop-off with AND-ing Inks

Using the basic invisible ink described above we note that contrast declines exponentially in the number of printers. In practice this greatly limits the number of printers that can participate and still produce a legible message. Other factors like image size, resolution and font play a role in legibility but in general we would not expect an obliviously printed document to be legible with more than about half a dozen printers.

We have discussed invisible ink in the context of a physical disjunction (i.e., an OR). In that setting a pixel will darken on activation if any of the shares contain invisible ink. However it seems invisible ink printing could offer other possibilities if the pigmentation reaction could be customized to realize a different logical construction. We briefly examine the properties that can be achieved if it were possible to formulate invisible inks that implement a physical conjunction (i.e., an AND). Chemically it seems possible such inks could be formulated; the basic invisible ink as described throughout this dissertation already constitutes a type of chemically-based conjunction: pigmentation occurs only when the invisible ink and activating substances are combined. Granted it would likely be a challenge to formulate conjunctive inks that were invisible for more than a small k . We are

not aware of the existence of such inks. It is worth noting, however, that if such inks *could* be formulated, they have the potential, at least in theory, to achieve optimal contrast (i.e., $\alpha = 1$) in the presence of arbitrarily many printers.

Definition 11.2. *A set of k inks are k -way conjunctive if, upon activation, a pixel darkens iff all k inks are physically present.*

We denote an n -VCX implemented with k such “AND-ing” inks as a (k, n) -VCXA. To create sub-pixel share matrix Φ in this setting we begin by constructing the $(n \times 2^{n-1})$ matrix S (refer to Definition 11.1) and then evenly segmenting it into $\frac{2^{n-1}}{k}$ sub-matrices of size $(n \times k)$. Each sub-matrix represents a sub-pixel, and each element in the sub-matrix instructs the printer whether to print the associated ink in that sub-pixel or not. Using this approach a (k, n) -VCXA has a contrast $\alpha = \frac{k}{2^{n-1}}$ (k is a power of 2 and the optimal contrast ratio remains $\alpha = 1$).

Example 11.2. *A $(4, 4)$ -VCXA: Let inputs $[a_1, a_2, a_3, a_4]$ and S be the same as in Example 11.1. The 4-way conjunctive inks are labeled A, B, C , and D . Each share instructs the printer which of the four inks to print in each of the two-subpixels. The shares are: $\Phi(1, 1) = [\{A, B, C\}, \{A\}]$, $\Phi(2, 0) = [\{C\}, \{A, C, D\}]$, $\Phi(3, 0) = [\{B\}, \{A, B, D\}]$, and $\Phi(4, 1) = [\{A\}, \{A, B, C\}]$. The conjunction of the shares produces $[\{A, B, C\}, \{A, B, C, D\}]$. Since the first sub-pixel will not contain the ink D when the shares are printed, it will never activate. The second sub-pixel will contain all four inks when printed and therefore will darken when activated. The pixel therefore will contain one white and one black sub-pixel which is visually interpreted as intended, i.e., a white pixel with contrast $\alpha = \frac{1}{2}$. By comparison with Example 11.1 the contrast is $4\times$ greater.*

11.6 Example Applications

11.6.1 Electronic Voting

Cryptographically verifiable electronic voting is a natural application for oblivious printing. In this setting voters receive a receipt of their ballot that allows them to confirm their vote was correctly counted, yet without revealing it to anyone. A vital requirement of any secret ballot election employing the receipt paradigm is that no single party, *including* the ballot printer(s), may gain an advantage in deducing how a voter voted.

Printing Verifiable Optical-scan Ballots. In Chapter 10 we presented a two-party approach to obviously printing ballots based on the two-party techniques of Chapter 9. Through the work presented in this chapter, we are now in a position to extend it to a fully

multi-party setting—a feature long realized in fully-electronic proposals (see the following chapter).

Multi-factor ballots for Internet Voting. Internet voting has been a recent and popular topic of interest. One successful open-source and cryptographically verifiable Internet voting platform is Helios.² Helios accepts encrypted votes (along with zero-knowledge proofs of validity), which are then homomorphically tallied [AMPQ09]. One fundamental and well-known limitation of this approach is that the voter’s computer must be trusted to construct the encrypted ballot and is vulnerable to viruses/malware. Using Protocol 11.5, encrypted Helios votes could be prepared on a voter’s behalf and mailed to them on an obviously printed ballot form. The voter would cast their vote by submitting the ciphertext corresponding to their candidate. Similarly, a verifiable Internet voting scheme due to Ryan and Teague [RT09b] proposes a multi-factor solution based on acknowledgment codes cards, which are mailed to the voter. The acknowledgment code cards contain secret information and so oblivious printing may be of use here also.

Coercion-resistant Internet Voting. Beginning with Juels et al. [JCJ05], work into coercion-resistant internet voting has attempted to extend privacy protection to voters, even when casting their ballots in an unsupervised environment. Clark and Hengartner [CH11] propose a coercion-resistant scheme based in part on an in-person registration protocol requiring voters to select secret passphrases and be able to (privately) compute randomized encryptions of them. Such passphrases and their encryptions could instead be pre-computed and obviously printed by a distributed election authority, potentially simplifying the in-person registration phase and simultaneously enforcing higher-entropy passphrases.

Electronic Cash. Bitcoin³ is an interesting recent proposal for digital currency. Transactions are timestamped and inserted into a common transaction history (known as a “block chain”) using a proof-of-work model. An account consists of a DSA keypair: a private signing key is used for sending funds and a public key is used for receiving them. A transaction consists of two components. The first component points to an earlier transaction in the block chain in which funds were sent to the account corresponding with the user’s public key (and for which the funds have not already been spent). The second component involves the user signing the transaction (which includes the destination account) using the private signing key. Typically these keys are stored on a user’s machine in a “wallet” file. One interesting alternative is Bitbills,⁴ a service which issues Bitcoins in

²<http://heliosvoting.org>

³<http://bitcoin.org>

⁴<http://bitbills.com>

physical form. A Bitbill consists of a plastic card (similar to a credit card) corresponding to a set amount of Bitcoins. The associated private signing key is printed on the card as a 2-D barcode and hidden under a tamper-evident/holographic covering. The funds can be redeemed by scanning the card with a smartphone.

Importantly, knowledge of the private signing key is necessary and sufficient to transfer funds and recent criminal activity has focused on stealing such keys from users' computers⁵ as well as from online Bitcoin bank accounts.⁶ Therefore any currency issuing service like Bitbills would have to be trusted never to redeem the cards it issues, and to prevent any private keys from falling into the hands of hackers. Oblivious printing could be used to create a *distributed* currency issuing service. With Protocol 11.5 adapted to an elliptic curve setting, keypairs could be generated and printed without any individual issuer knowing the private key thereby enforcing that only the cardholder can redeem the funds.

11.7 Security Analysis

We briefly sketch some of the security properties of our system. For space reasons we limit our discussion to Protocol 11.3 (i.e., Subprotocols 11.3(a) and 11.3(b)).

Cryptographic Security. Informally there are two security properties we seek for the cryptographic component of the protocol. One is *integrity*: a printer should be convinced that the combined shares depict an image of the (encrypted) input. The other property is *secrecy*: an adversary in collusion with a subset of printers should not be able to determine the input.

We assume the commitment function is non-malleable, hiding and binding. The assumptions regarding encryption are stated in Section 11.3.2. The completeness, soundness and secrecy of Sub-protocol 11.3(a) follow directly from [JJ00, JJR02]. If the printers follow Sub-protocol 11.3(b) they will always produce a finalization layer that, when XOR-ed with the individual shares, recovers the input. Secrecy of the commitments and encryptions follow from the assumptions. Secrecy of the decrypted finalization layer follows if one or more printers select random bits. Soundness is probabilistic and follows from the cut-and-choose proof. The independence of the random bits is enforced by the non-malleable commitment function. Correct computation of the homomorphic XOR is established by the cut-and-choose proof when printers reveal their commitments and the random factors used to compute the XOR.

⁵<http://bitcointalk.org/index.php?topic=16457.0>

⁶<http://mybitcoin.com/archives/20110804.txt>

Physical Security. For simplicity we proceed with our discussion of physical security in a setting in which the printers receive their shares from a trusted dealer through a private and authenticated channel. In the physical setting we seek two security properties. One is *integrity*: a printer should be convinced that the combined printed shares match the combined received shares. The other property is *tamper evidence* which is closely related to secrecy: an adversary should not be able to determine the output of the protocol without corrupting all printers or tampering with the document, which will then be evident.

We assume the invisible ink can only be read in its activated state and that activated ink is plainly evident. We assume that a sheet of paper can be authenticated. Completeness of Sub-protocol 11.3(b) is self-evident. Secrecy of the shares follows from the properties of an n -VCX. If a printer attempts to read the document by activating the ink it will be evident following from the assumptions of the invisible ink. If a printer attempts to replace a valid document with a fake it will be evident following the assumptions regarding document authentication. Soundness is probabilistic and follows the cut-and-choose proof. If a printer prints nothing in a sub-pixel where it was to print invisible ink, it will either be covered by invisible ink from another share, and does not alter the intended outcome, or, it will not be covered by another share in which case it will be detectable by the cut-and-choose and attributable by examining the electronic shares. If a printer prints invisible ink in a sub-pixel where it was to print nothing, it will be detected similarly but is not attributable.

It is important to note that nothing fundamentally prevents an adversary in physical possession a document from activating the ink and reading its contents. The severity of this threat will depend greatly on the use case. For example if the document contains a *unique secret*, additional physical security measures are necessary to protect document secrecy. Alternatively if the document contains an *arbitrary secret* (e.g., a new password), it may suffice for the recipient of a tampered document to simply request it be invalidated and a new one be issued.

11.8 Concluding Remarks

In this chapter we generalized oblivious printing to a fully multi-party setting. We presented three protocols: a generic protocol for obviously printing an encrypted plaintext, a protocol with improved contrast for obviously printing a random message, and third protocol to generate and obviously print a DSA/Elgamal keypair. We then proposed a contrast optimization based on the AND-ing invisible inks and provided some example applications for electronic voting and digital cash.

While we believe our experience with invisible ink printing (cf. Chapters 4 and 5) suggests that invisible ink *overprinting* will provide the necessary properties, an obvious

direction for future work is to test this hypothesis in practice. Other important questions will include finding realistic upper bounds on the number of participants, and on sub-pixel resolution producing legible results.

Chapter 12

Putting it all Together

This chapter is adapted from published work supervised by Urs Hengartner [EH12a].

12.1 Introductory Remarks

In this Chapter we present HOVER (*Hash-Only VERification*), a sketch of a protocol for a trustworthy voting system that combines the research results presented in the previous chapters. At the core of HOVER is Scantegrity. We borrow aspects from Eperio to simplify audit procedures, such as through verification in a spreadsheet. We apply oblivious printing and the related techniques of Scantegrity 3-D to distribute trust among multiple election trustees without increasing the technical requirements of the election audit. Finally, we round out the system with a proposal for hash-only verification, which we believe can be a tool for educating less-technical audiences about the merits of cryptographically verifiable elections.

Contributions. The contributions of this chapter are summarized as follows:

- A proposal for hash-only verification.
- HOVER: a cryptographically verifiable optical-scan system with:
 - A conventionally marked paper optical-scan ballot, as developed in Part I,
 - A simplified audit procedure based on the results of Part II,
 - Multi-party distribution of trust, as developed in Part III

12.2 Hash-only Verification

In this chapter we propose an end-to-end proof based around a cryptographic hash function. Why hashing? To facilitate our discussion, our rationale is given from the perspective of two groups: a non-expert technical audience with a very basic background in cryptography but with a potential interest in election auditing, and a general audience without *any* background in cryptography but with an interest in learning about the underlying principles.

To a non-expert technical audience interested in performing an audit, hashing is one of the most commonly encountered cryptographic primitives. For example, anyone who has downloaded open source software is likely to have directly encountered a hash. Hashing is also one of the few primitives that is commonly found in the standard libraries of (or at least widely available for) many programming languages. A file hash can even be computed directly from a UNIX based command line (e.g., Linux and Mac OS), and numerous free websites and utilities exist for users of other operating systems. By comparison, many of the cryptographic primitives favored in the literature (e.g., homomorphic encryption and zero-knowledge proofs) do not have widely available implementations, often requiring that they be implemented from scratch.

To a general audience interested in understanding the cryptographic properties, hashing provides an intuitive parallel with physical fingerprints. Such a metaphor seems potentially useful given the popularity of prime-time crime dramas. For example, a viewer of such a show might conceivably understand that police could not easily identify a fingerprint that was not already in their criminal database (preimage resistance). Such a person might also understand that finding two people with matching fingerprints is unlikely (collision resistance) and that finding another person whose fingerprints match their own would be even more unlikely (second preimage resistance). Although this is only an imperfect analogy, we can reasonably expect to face fewer obstacles connecting a new concept to an existing and well formed mental model. By comparison, many of the primitives favored in the literature do not have nearly as strong a connection to an existing model. For example, semantically-secure additively homomorphic public-key encryption would require conveying three novel concepts instead one: randomized encryption, adding “under the covers”, and asymmetric keys, none of which have as strong a physical analogy.

12.2.1 A Special Purpose Hash-based Commitment

To the end of a hash-only verifiable election scheme, we would like to employ a hash function (under specific assumptions) as a computationally hiding and computationally binding commitment scheme. Cryptographic commitments are at the core of a number of

trustworthy voting protocols such as Scantegrity and Eperio. A number of commitment schemes exist in the literature, such as the unconditionally hiding commitment scheme due to Pedersen [Ped92]. In this application we are interested in using hashing for its relative technical and conceptual simplicity.

Given a collision-resistant hash function \mathcal{H} and a message m , however, $\mathcal{H}(m)$ is not a commitment to m in general. For one thing, hashing a message is not computationally hiding when m is not chosen from a sufficiently large space. For example, committing to a single bit $b \in \{0, 1\}$ by posting $c = \mathcal{H}(b)$ is trivially opened by checking which of $\mathcal{H}(0)$ and $\mathcal{H}(1)$ produce c . Under certain circumstances, however, such as when the message is itself a sufficiently large random factor, a hash could potentially function as part of an *application-specific* commitment. Although a commitment function based on a collision-resistant hash function can be considered a “false solution” in general, we believe it may be suitable for our specific needs.

Given a collision-resistant hash function $\mathcal{H} : \{0, 1\}^* \rightarrow \{0, 1\}^\ell$ and an efficient description of permutation space Π , for $|\Pi| \geq 2^\ell$, we define an application-specific commitment function as follows: $\text{Comm}(\pi)$ takes permutation $\pi \in_R \Pi$ and produces a commitment $c = \mathcal{H}(\pi)$. $\text{Open}(c, \pi')$ takes a commitment c and an asserted permutation $\pi' \in \Pi$ and returns `accept` iff $\mathcal{H}(\pi') = c$.

For $\pi, \pi' \in \Pi$, Comm is binding as long as an adversary cannot find a *collision*, i.e., $\mathcal{H}(\pi) = \mathcal{H}(\pi')$ for any $\pi \neq \pi'$, or, find a *second preimage*, i.e., given π find a $\pi' \neq \pi$ such that $\mathcal{H}(\pi) = \mathcal{H}(\pi')$. We assume it is computationally infeasible to find valid collisions as part of the collision-resistant properties of \mathcal{H} . We assume it is computationally infeasible to find valid second preimages as part of the one-way properties of \mathcal{H} (discussed below). Comm is hiding as long as an adversary cannot exhaustively search the message space, or invert the hash function, i.e., given c , it should be hard to find a $\pi \in \Pi$ such that $\mathcal{H}(\pi) = c$. We assume π is chosen uniformly at random from Π and that $|\Pi|$ is sufficiently large that exhaustively searching the message space is computationally infeasible. We assume it is computationally infeasible to invert \mathcal{H} as part of the one-way properties of \mathcal{H} .

An important question is whether collision-resistance implies one-wayness. Bellare and Rogaway [BR05] show that an adversary with the ability to attack the one-wayness of a hash function $\mathcal{H} : D \rightarrow R$ has negligible advantage over an adversary with the ability to attack the collision-resistance of \mathcal{H} in the case where $|R|/|D|$ is also negligible. For typical real-world election parameters, we show a collision-resistant hash function will imply one-wayness for our purposes.

In our particular application the message we are committing to (i.e., the hash image) is a random permutation of a list of the mark state (i.e., marked or unmarked) of each of the optical-scan ovals on each of the ballots cast in the election. In an election involving c candidates and v voters, this list consists of cv elements, of which there are $(cv)!$ possible

permutations. As an example, for a small election of $c = 2$, $v = 50$ voters, and a hash function with an output of length of $\ell = 256$ bits, we have $|D| = 100!$, $|R| = 2^{256}$, and $|R|/|D| \approx 2^{-269}$, which is negligible in ℓ .¹

As Halevi and Micali [HM96] warn, however, despite \mathcal{H} being collision resistant and one-way, there are no guarantees about the feasibility of computing partial information about m given $\mathcal{H}(m)$. Their point is that to ensure the commitment is hiding, we must require additional properties of \mathcal{H} . For the sake of this article we exclude the threat of partial information leakage, for example, by modeling \mathcal{H} as a random function. This is somewhat in contradiction to our real-world setting, so it is worth briefly outlining potential consequences when there exists an adversary who is able to extract a small number of bits from m given $H(m)$. Although we believe that the overall threat to voter privacy is low, we note that proving this would be complex and would require specific assumptions about the distribution of the leakage for a given hash function. Still, we note aspects of our system offer an inherent degree of fault tolerance. For example, a leakage of g bits is not sufficient to fully determine any single input/output pair of a permutation so long as $\log_2(cv!)/cv > g$. Thus leaking a small number of bits (e.g., 1–2) would never be sufficient to fully determine any single input/output pairing of a permutation for most election sizes. Intersection attacks combining partial leakages across multiple proof instances, however, remain a possibility. Based on previous (unpublished) analysis of Scantegrity however, we contend this threat is fairly minimal when considering small leakages.

12.3 Basics and Election Setup

HOVER is a paper optical-scan voting system largely related to the Scantegrity system [CCC⁺08, CCC⁺09]. For simplicity, we will describe the election correctness proof as being generated by a single trusted authority. Although a trusted authority cannot create a false (but accepting) proof, like in Scantegrity, it does receive sufficient information to link individual voters to their voting intentions. In the following section, therefore, we outline how recent advances in secure document printing [EH12b] can be used to distribute trust among multiple authorities.

A single election trustee \mathcal{T} initializes an election by establishing a public append-only bulletin board \mathcal{BB} . For an election involving c candidates and v voters, \mathcal{T} defines a permutation space Π consisting of all possible permutations of cv elements. \mathcal{T} posts c, v , an efficient description of Π , a description of a collision resistant hash function \mathcal{H} , a canonical list of candidate names N , and an alphabet of c valid/possible confirmation codes D to \mathcal{BB} . Note in practice v is several times larger than the actual number of registered voters to allow for some ballots to be spoiled and/or audited.

¹We say a function f is *negligible* in positive integer k if $f(k) < 1/\text{poly}(k)$.

\mathcal{T} generates v ballots, each containing c unique tuples $\{s, d, n\}$ associated with a specific optical scan oval on the ballot. This tuple consists of a ballot specific serial number $s \in \{1, \dots, v\}$, a code letter $d \in D$ assigned randomly without replacement, and the associated candidate name $n \in N$. Note that the association between a given code d and candidate name n is independent and random across ballots. All v ballot tuples form a master ballot table $B = \{B_s, B_d, B_n\}$. B is given to a trusted printer who prints each of the v ballots.

For each proof instance $i = 1 \dots p$, \mathcal{T} chooses two random permutations $\rho_i, \sigma_i \in_R \Pi$ and computes shuffled candidate list $B_{n_i} = \rho_i \circ \sigma_i(B_n)$. \mathcal{T} posts serial number lists B_s , code list B_d , each of the p shuffled candidate lists B_{n_i} , and commitments to the associated random permutations, $\mathcal{H}(\rho_i)$ and $\mathcal{H}(\sigma_i)$, to \mathcal{BB} .

To vote, a voter marks the optical-scan oval appearing beside their chosen candidate's name \hat{n} . A voter creates a receipt $r = \{\hat{s}, \hat{d}\}$ by writing down the serial number \hat{s} appearing on their ballot and the code letter \hat{d} appearing beside the marked oval. A ballot, its receipt, and the associated entry in the bulletin board are depicted in Figure 12.1.

Because of the unsupervised nature of receipt creation, it is possible that disputes can arise between a voter and \mathcal{T} over which code was marked. In its basic form, HOVER utilizes the in-person dispute resolution procedure from Scantegrity (cf. Chapter 3), which uses a non-cryptographic cut-and-choose proof befitting with our design goals, although as discussed in previous chapters, does not scale well and is cumbersome to administer in practice. Scantegrity II informational dispute resolution (cf. Chapter 4) is more practical, though as a drawback, it is conceptually more complicated and must be adapted to a distributed trust setting such as discussed in Chapter 10. This comes, however, at the conceptual cost of a zero-knowledge proof. Designing a dispute resolution procedure that is scalable, uses an economy of cryptography, and yet is secure in a distributed trust setting, is a matter we leave for future work.

12.3.1 Linkage Lists

As a simplification over Scantegrity, HOVER commits to the full specification of random permutations σ_i and ρ_i , whereas Scantegrity publishes separate commitments to each individual input/output pair. Committing to an entire permutation not only allows us to use the simpler commitment scheme described above, it also results in substantially fewer cryptographic operations overall: $2p$ commitments in the former case, and $2pcv$ commitments in the latter case.

An important part of a trustworthy election is the ability for a voter to receive assurance that the code/candidate association appearing on a given paper ballot is faithfully represented in the bulletin board tables. In a Scantegrity election, a voter (or some other

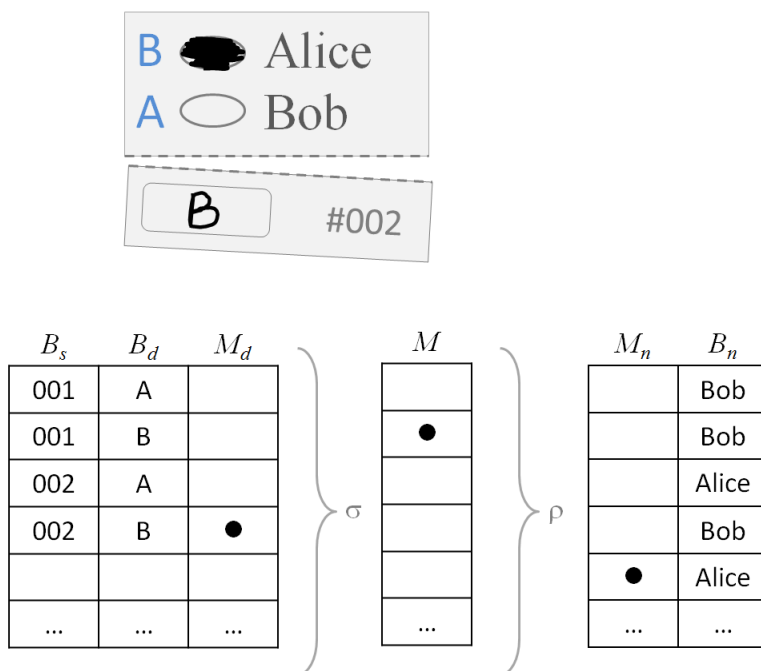


Figure 12.1: Top: A marked ballot and receipt. Bottom: A bulletin board showing receipt and candidate information including serial numbers B_s , confirmation codes B_d , and the mark state of the associated optical scan ovals M_d (left). A list of candidate names B_n and the associated mark states M_n (left) is dissociated from the corresponding entries in B_s , B_d , and M_d by a secret shuffle. The shuffle is expressed as the composition of two independent random permutations σ and ρ —one may later be challenged by the public to be revealed as part of a cut-and-choose proof of correctness. Additional independent shufflings can be generated to increase soundness.

designated 3rd party auditor) may challenge \mathcal{T} to prove the correct printing of any ballot. For each element in serial number list B_s and code list B_d associated with serial number \hat{s} of the challenged ballot, \mathcal{T} discloses each associated element in each of the p candidate lists B_{c_i} by opening the commitments to the associated input/output pairs in ρ_i and σ_i ($2pc$ openings in total). Ballot \hat{s} is then considered spoiled and may not be voted on; a poll worker marks the paper ballot accordingly, and a flag is placed beside the associated entries in the bulletin board.

Because HOVER commits to permutations as a whole, it cannot reveal individual input/output pairs directly. To address this, we make use of the “linkage list” construction used by Eperio (cf. Chapter 7). Instead of proving code/candidate links by directly opening the affected commitments, \mathcal{T} merely asserts the links in a linkage list LL . The

correctness of these assertions can later be confirmed (with high probability) as part of the post-election audit. Although we feel the linkage list approach provides a much simpler verification process through full-permutation commitments, a major drawback is that faults in the printing are only discovered after the election.

An important question is whether our hash based commitment securely composes with the use of a linkage list. Through the linkage list, we are intentionally disclosing partial information about permutations ρ_i and σ_i . This by itself is not an issue: the number of unaudited ballots is always at least as large as the total number of voters, meaning the effective permutation space $\Pi_{ef} \subseteq \Pi$ will still be large despite partial disclosures due to the print audit. It could be the case, however, that a real-world cryptographic hash function might leak additional bits as a consequence of this partial disclosure. Of course, if \mathcal{H} is a random function, information revealed by the print audit cannot be used to compute unknown bits.

12.3.2 Election Challenges and Audit

After the election, \mathcal{T} uses the data collected by the optical scanners during the election to create a mark state list M_d of cast ballots. $M_d(i) \leftarrow 1$ iff the confirmation code $B_d(i)$ on ballot $B_s(i)$ was recorded as having been marked, or $M_d(i) \leftarrow 0$ if it was unmarked. $M_d(i) \leftarrow \emptyset$ if the ballot was spoiled and/or print audited. \mathcal{T} posts M_d to \mathcal{BB} . Finally, for each proof instance $i = 1 \dots p$, \mathcal{T} creates shuffled marks tables $M_i = \sigma_i(M_d)$ and $M_{c_i} = \rho_i(M_i)$, and posts these lists to \mathcal{BB} . A shuffled candidate name list B_{n_i} and the associated marks list M_{n_i} are sufficient to compute an election tally. Anyone can check that each of the p instances produce the same election result.

Audit Challenges. For each proof instance $i = 1 \dots p$ the public will collectively issue a challenge to \mathcal{T} to reveal either σ_i or ρ_i . \mathcal{T} responds in kind by opening the associated commitments. There are a number of ways to produce challenge bits, the essential property is that \mathcal{T} must not be able to predict the outcome with an advantage over a random guess. One potentially suitable source of entropy for small scale elections will be the upcoming random beacon currently being implemented by the National Institute of Standards and Technology (NIST) [FIP11]. The NIST beacon will draw on hardware based entropy sources, offering the public digitally signed random bit strings in a persistent on-line database. Although in general the NIST beacon will be agnostic about what its outputs are being used for, it still represents a trusted component in this context, which may not be suitable for larger elections. An alternative is randomness extracted from financial data—a method we used in the 2009 Scantegrity election in Takoma Park. Clark and Hengartner provide an analysis of this approach as well as suggestions for securely combining the outputs of multiple independent sources [CH10].

Election Audit. A voter can use their receipt $r = \{s, \hat{d}\}$ to check that, for each row $s \in B_s$, all the associated mark states in M_d are unmarked, except for the one associated with their recorded confirmation code \hat{d} in B_d . Anyone can check that the mark states were correctly propagated between the receipt and candidate tables by checking against the opened permutations, i.e., by checking that $M_i = \sigma_i(M_d)$ or that $M_{n_i} = \rho_i(M_i)$. The printing of challenged ballots is verified first by checking that the code/candidate associations indicated in the linkage list match those indicated on the corresponding paper ballot. Finally, the partial permutations in the linkage list are checked against the opened permutations.

12.3.3 Verification in a Spreadsheet

A common roadblock faced by many real-world implementations of cryptographic election verification is the relative complexity of the software. Unlike conventional software, which can effectively function as a black box, election verification software must be developed in a way that is functionally and conceptually transparent. Consistent with our design goals, the election protocol should allow the verification software to be written in an economy of lines of code (or perhaps more aptly put: “an economy of attention”²). Similarly, due to the nature of highly specialized cryptographic components necessary to many crypto voting protocols, the option to use existing software often does not exist.

Borrowing from the design goals set out in Eperio, we believe a verifier should be able to audit a cryptographic election using existing software as much as possible, or, looked at from the other direction, be required to write as few lines of software as possible. With Eperio we presented a small audit tool written in 50 lines of Python. As an alternative—one requiring *no* new lines of code—we presented an example in which an election could be manually audited using a desktop spreadsheet. Although a manual audit would be tedious for large scale elections, the widespread use of spreadsheets does present an interesting opportunity for explaining the election audit in familiar terms. In this way the audit can be communicated as a series of basic spreadsheet operations such as copying and pasting columns of data, and simple commands like `sort` and `find`.

To facilitate spreadsheet verification, \mathcal{T} encodes each of the ballot and mark lists (B_s, B_d , etc) as a comma-separated values (CSV) file. The random permutations ρ_i and σ_i , are expressed as lists of shuffled integers $1 \dots cv$, also encoded as CSV files. Hashes of permutation files are posted as commitments. \mathcal{T} responds to the post-election challenges by posting the relevant permutation files. The verifier first checks that the asserted file is valid, i.e., it is in the correct encoding, and that the file only contains a valid shuffle of the integers $1 \dots cv$. The verifier then executes a command line file hash utility and compares

²Quote thanks to Ian Goldberg.

the result to the posted commitment. For example, if \mathcal{H} is SHA-256, the verifier could run the file hashing utility `sha256sum`. Optionally these tasks can be automated using a simple shell script. Once satisfied the commitments are correct, the verifier opens each of the files in a spreadsheet, e.g., OpenOffice Calc. The audit steps are followed mostly as above. To perform a permutation, e.g., to check if $M_i = \sigma_i(M_d)$, the verifier copies the list of shuffled integers from the σ_i worksheet and pastes them into the M_d sheet. The verifier then sorts the sheet using the shuffled integers as the sorting key. The verifier pastes M_d (now shuffled) into the M_i worksheet and tests the lists for equality. The linkage lists are checked similarly. A number of built-in spreadsheet commands exist for efficiently performing such comparisons, and audit can be automated by a spreadsheet macro.

12.4 Distributing Trust

So far \mathcal{T} has been described as a single trusted entity, i.e., one who knows the associations between receipts and ballots (i.e., between voters and cleartext votes). Scantegrity, as an example, makes extensive use of trusted entities and hardware: the optical scanners, the ballot printers and even the poll workers gain access to receipt/vote combinations. Because of the physical nature of the paper optical scan ballot, however, distributing trust in this setting has proven to be a challenge. We use the secure multi-party oblivious printing protocol of Chapter 11 to overcome the need for a trusted printer.

Let \mathcal{T} be replaced by a collection of t trustees $\mathcal{T}_1 \dots \mathcal{T}_t$. First, the trustees run an oblivious printing protocol to randomly generate and obviously print the confirmation codes on each of the v ballots. The output of this protocol includes the obviously printed ballots, as well as a vector of encryptions of each of the code/candidate associations for each ballot. These encryptions are made using a semantically secure public-key encryption scheme for which the decryption key is distributed among the trustees.

For each proof instance $i = 1 \dots p$, each of the $j = 1 \dots t$ trustees separately generates and posts commitments to 2 random permutations ρ_i^j, σ_i^j . Next, the trustees compute the shuffled candidate name lists B_{n_i} using a reencryption mixnet taking the encrypted candidate names as input and decrypting the shuffled result. The overall permutation on the candidate list B_{n_i} is thus $\rho_i^t \circ \rho_i^{t-1} \circ \dots \circ \sigma_i^2 \circ \sigma_i^1$. The trustees similarly apply these permutations when populating the marks lists after the election. During the audit challenge, each trustee \mathcal{T}_j opens the commitment to the requested permutation, either σ_i^j or ρ_i^j . The audit proceeds as above except with the verifiers checking a composition of the trustees' permutations, i.e., by checking either $M_i = \sigma_i^t \circ \dots \circ \sigma_i^1(M_d)$ or $M_{n_i} = \rho_i^t \circ \dots \circ \rho_i^1(M_i)$.

This approach allows HOVER to be run with a distributed set of trustees and without a trusted printer. However this approach does not offer *fully* distributed trust: the confir-

mation code associated with the chosen candidate is still revealed to the optical scanner (and potentially the poll workers) when a ballot is cast. Following the approach taken in Chapter 10, we can define two independent serial numbers: one for the ballot, and one for the receipt. We discuss the additional procedures necessary such that no single entity in the election, including the voter, ever sees both serial numbers. Finally, although the verifier must verify the openings of t times as many commitments, at a conceptual level the audit procedure is essentially the same as in the single-party case.

12.5 Concluding Remarks

Returning to our design criteria we consider how HOVER meets the design goals. In terms of cryptographic verifiability, HOVER provides computationally sound election audits similar to those proposed in Scantegrity and Eperio. With regard to distribution of trust, HOVER makes use of the new oblivious printing paradigm to offer optical-scan paper ballots allowing a voter to cast a vote and construct a receipt, without anyone else finding out who they voted for. In terms of the latter two properties—a usable voting interface, and a conceptually simple verification procedure—it is harder to ascertain how HOVER fares without doing a usability study. HOVER offers voters and election officials a familiar paper optical-scan ballot. Our voter poll in the 2009 Takoma Park election (see Chapter 5) suggested voters and election officials found Scantegrity II ballots reasonably intuitive to use. It is unclear, however, how these perceptions would be altered by the changes introduced by the obliviously printed elements, and would be a question for future work. Until we can satisfactorily address the threat of coercion in internet voting, however, we can expect paper ballot optical-scan to be the dominant approach to cryptographically verifiable voting.

Finally with regard to a simple verification procedure, we proposed a simple verification procedure based around a hash-based commitment. We argue that, at a conceptual level, hashing has a closer connection to an everyday paradigm than many of the primitives found in the voting literature. At a technical level, we explain how hash-based commitments can lead to smaller code sizes of audit software, even giving an example of how it might be performed in a desktop spreadsheet. Ultimately it is our hope HOVER will be a stepping stone toward greater acceptance and awareness among non-cryptographers of the merits of cryptographic election verification.

Chapter 13

Discussion, Future Work, and Conclusion

Always vote for principle, though you may vote alone, and you may cherish the sweetest reflection that your vote is never lost.

John Quincy Adams

This chapter contains excerpts from published work [EH12a].

13.1 Discussion

The design of cryptographically end-to-end verifiable elections is both a technical and social challenge. At a technical level, cryptography can make elections verifiable in a way that fundamentally exceeds conventional approaches. It offers each voter a means to check that *their* ballot was counted correctly, without revealing it to anyone. The trustworthiness of such elections is the product of statements proven directly about the election results, not of individual voting machines, or the software they supposedly run.

Through this dissertation we have attempted to innovate the technical in the hope of innovating the social, so as to bring cryptographic election verification closer to the mainstream (i.e., what cryptographers call the “real world”). One possibly for gauging how far we have come is Stu Feldman’s roadmap for technical maturity (as quoted in [Gee01]) consisting of the following milestones:

1. You have a good idea,
2. You can make your idea work,
3. You can convince a friend to try it,
4. People stop asking why you are doing it,
5. Other people are asked why they are not doing it.

Beginning with Chaum and Benaloh in the 1980's onward to recent times, we see the foundations of our "good idea." Where has this dissertation taken us from there? How have we done with respect to our objectives (as stated in Section 1.2)?

Objective I: Improve Real-world Practicality for Voters and Election Officials.

With Chapters 3 and 4 we began with our good idea: a cryptographically verifiable voting system based on a conventional optical-scan paper ballot. In Chapter 5 we made our idea work, and convinced a friend¹ (Takoma Park) to try it.

What we found was, in stark contrast to our early attempts with Punchscan, voters found the ballot intuitive to mark, and appreciated the fact they could create a ballot receipt (but didn't *have to*). The Takoma Park election officials, for their part, were able to administer the election without major hindrance. Although only a formal usability study could confirm, our experiences running elections of this kind indicate that our goal of improving real-world practicality for voters and election officials was met.

Objective II: Improve Real-world Practicality for Election Verifiers. With Chapters 6, 7, and 8 we pursued the challenge of getting people to stop asking why we are doing cryptographic election verification.

Aperio offers individuals interested in the procedures of cryptographic election verification to learn about it in a non-technical way. Eperio and CommitCoin simplifies procedural and software requirements for auditors with some technical background. Eperio vastly diminishes the size of audit software, as well as modularizes the role of cryptography, with Aperio forming a pedagogical stepping stone. CommitCoin removes the requirement that potential election verifiers become involved prior to the election. Through this work, election verifiers have been given choice in audit software, a more flexible schedule, and a gentler learning curve.

¹Feldman's original quote was "convince a *gullible* friend to try it." Takoma Park, however, spent over a year investigating our proposal before voting to adopt it.

Objective III: Distribute Trust in Paper Ballot Elections. With Chapters 9, 10, and 11 we pursued the challenge of distributing trust of printing cryptographically verifiable paper optical-scan ballots. Although this line of research does not fit in to Feldman’s progression directly, it does so indirectly—strong protections to ballot secrecy, after all, should be a pre-condition for adoption.

Throughout this dissertation we have argued the practical merits of optical-scan relative to fully electronic schemes. Prior to this work, however, the only options for trustworthy optical-scan voting were to either accept weaker assurances of ballot secrecy due to reliance on trusted hardware/entities, or, to burden voters with unusual/novel ballot styles and marking procedures. Following this work, we can offer voters a paper optical-scan ballot *and* better protections on ballot secrecy through a distribution of trust.

13.2 Future Work

The final stage of Feldman’s timeline—other people are asked why they are not running cryptographically verifiable elections—remains to be seen. Perhaps we may not witness this in the foreseeable future: the necessary legal and policy framework for large scale deployment will take time, even if widespread demand for the technology can eventually be fostered. There are, however, a number of technical and usability improvements that could be investigated in the meantime:

- Future work arising from Part I:
 - **Usability of successive elections:** with any new technology there is an associated learning curve. It would be important to understand how the perceived and actual usability of Scantegrity improves through exposure to successive elections. Preliminary data of the 2011 Takoma Park election (the second successive use of Scantegrity) suggests that voters may be more confident with procedures the second time around, although a formal user-study to confirm this will be an important next step.
 - **Error detection success rates:** a major issue central to cryptographic verification is just how likely the general public is to detect election audit errors when they occur. A user study (in the context of a mock election) would be useful for understanding how verifiers might interpret errors when they occur, as well as what action they might take in communicating it to the appropriate authorities,
 - **Error attribution:** if an error is detected in an election, it would be vitally important to be able to distinguish between a benign procedural/technical failure and a malicious/fraudulent act. Further, improvements to the underlying

cryptography might be a useful direction for future work for the purposes of attributing the error to a particular hardware device, polling station, poll worker, ballot printer, voter, or election official, etc.

- **Fault recovery policies:** our research has thus far focused on detecting election-related errors, but the field has thus far made little progress in creating policies for handling interpreting errors and recovering from them. In what circumstance should an election found to contain errors be re-run? In what circumstance should the results be upheld? These questions will prove critical in the decision making process viz. adoption and deployment.
 - **Fault recovery technologies:** one of the most promising and unexplored areas for future work is the development of cryptographic means to reliably and convincingly recovering from faults detected by the audits, specifically without resorting to re-running the election.
 - **Invisible ink printing at scale:** one area for future work would be refinement of the invisible ink printing process. As a proof of concept, ballot printing for the 2009 Takoma Park election was successful, however, the ink chemistry at the time caused premature printer failure resulting in high production costs. Printing with commercial inkjet printers is also slow, labor intensive, and costly. Although we made improvements in the inks for the 2011 election, printer life span and production output must be improved before elections on the scale of a medium sized cities, or larger, will be feasible and cost effective. One possible area for exploration would be to develop an invisible toner (as opposed to ink), and integrate it into a high-speed digital industrial press. Our private conversations with a large provider of such technology suggests that this may be possible.
- Future work arising from Part II:
 - **Security of commitments:** the security of commitment schemes based on simple primitives such as block-cipher encryption or hashes should be studied in greater detail, with associated security proofs made in stronger models.
 - **Usability of verification:** the usability of the verification procedure is an important and unexplored area. It would be interesting to establish success rates for verification using a range of verification tools (small script, spreadsheet, etc.).
 - **Testing understandability:** it would be very important to the potential success of cryptographic election verification to gather qualitative data with regard to how understandable the principles are to different demographics, such as the general public, election officials, non-expert technical audiences, etc., as well as

to understand how much education is needed to bring average users from each demographic to the point where they would find a successful election audit to be convincing.

- Future work arising from Part III:
 - **Feasibility of invisible ink overprinting:** our experience with single layer invisible ink printing (as in Scantegrity) suggests that multiple layer overprinting will likely perform as required by the oblivious printing paradigm. This has yet to be confirmed in practice, however, and will be an important next step. One important question would be to place a practical upper bound on how many layers could be combined. A related question pertains to the practical limits of image registration, i.e., whether commercial printers can reliably be made to print in exactly the same spot on a sheet of paper twice,
 - **Upper bound on participants:** our use of visual cryptography results in contrast levels that decline exponentially in the number of participants. It would be important to study the legibility of obviously printed documents, varying the number of printers, sub-pixel resolution, absolute print size, font, etc. Early experiments suggest three parties is a minimal lower bound (i.e., more than three parties may be possible),
 - **Attribution:** with the oblivious printing protocol presented in this dissertation, it is generally possible to catch attempts to manipulate the message result; however, for the most part, it is not possible to attribute such malicious behavior to a particular printer. Future work should focus on developing efficient modifications to the cut-and-choose printing audit so as to be able to attribute faults to the responsible printers, allowing the protocol to be rerun without the offending parties present,
 - **Usability of Scantegrity 3-D ballots:** Several potential usability issues would need to be established with regard to the privacy mechanisms of obviously printed/Scantegrity 3-D ballots. Three areas to investigate would be the secure transportation/storage of ballots between the printers and the polling place, proper handling of ballots by poll workers when issuing blank ballots and scanning marked ballots, and finally, how well voters will reliably understand how to protect their ballot choices (e.g., waiting until the disappearing ink has completed reacting).

13.3 Final Remarks

Recall the old engineering adage: “fast, good, cheap—pick any two.” It is a testament to the trade offs inherent to any design project. Over the last 30 years of research into cryptographically end-to-end verifiable elections, the trade offs have primarily been between cryptographic verifiability, distribution of trust, a usable voting interface, and a conceptually simple verification procedure. Cryptographers have tended to optimize on the former two, whereas election officials and voters have championed the latter two. In our view, compromise across each of these properties is necessary to advance the cause of trustworthy elections.

In this dissertation, we have attempted to build a bridge between the priorities of both groups—from blackboard to election board. Our experience has been that, at times, neither group has felt entirely satisfied. Still, we take heart in the fact that a successful negotiation tends to proceed in such a manner. We still believe cryptographic verification represents the future of elections, and the research presented in this dissertation was made in support of this belief. By making cryptographic end-to-end election verification more accessible to the public, we contend that this research is an important step toward safeguarding democracy in the digital age.

References

- [ABSS96] G. Ateniese, C. Blundo, A. D. Santis, and D. R. Stinson. Visual Cryptography for General Access Structures. *Information and Computation*, 129:86–106, 1996.
- [ACC⁺08] A. Aviv, P. Cerny, S. Clark, E. Cronin, G. Shah, M. Sherr, and M. Blaze. Security Evaluation of ES&S Voting Machines and Election Management System. In *EVT*, pages 1–13, 2008.
- [ACG06] R. Araujo, R. F. Custodio, and J. v. d. Graaf. A Verifiable Voting Protocol based on Farnel. In *WOTE*, 2006.
- [Adi08] B. Adida. Helios: Web-based Open-audit Voting. In *USENIX Security Symposium*, pages 335–348, 2008.
- [AFT07] R. Araujo, S. Foulle, and J. Traore. A Practical and Secure Coercion-resistant Scheme for Remote Elections. In *Frontiers of Electronic Voting*, 2007.
- [AFT10] R. Araujo, S. Foulle, and J. Traore. A Practical and Secure Coercion-resistant Scheme for Internet Voting. *Toward Trustworthy Elections*, LNCS 6000, 2010.
- [AGH⁺09] A. W. Appel, M. Ginsburg, H. Hursti, B. W. Kernighan, C. D. Richards, G. Tan, and P. Venetis. The New Jersey Voting-machine Lawsuit and the AVC Advantage DRE Voting Machine. In *EVT/WOTE*, 2009.
- [AJL04] A. Ambainis, M. Jakobsson, and H. Lipmaa. Cryptographic Randomized Response Techniques. *PKC*, 2004.
- [AMPQ09] B. Adida, O. d. Marneffe, O. Pereira, and J.-J. Quisquater. Electing a University President using Open-Audit Voting: Analysis of real-world use of Helios. In *EVT/WOTE*, 2009.
- [AN06] B. Adida and C. A. Neff. Ballot Casting Assurance. In *EVT*, 2006.

- [AN09] B. Adida and C. A. Neff. Efficient Receipt-Free Ballot Casting Resistant to Covert Channels. In *EVT/WOTE*, 2009.
- [ANL00] T. Aura, P. Nikander, and J. Leiwo. DoS-resistant Authentication with Client Puzzles. In *Security Protocols*, 2000.
- [APR07] J. A. Aslam, R. A. Popa, and R. L. Rivest. On Estimating the Size and Confidence of a Statistical Audit. In *EVT*, pages 8–8, Berkeley, CA, USA, 2007. USENIX Association.
- [AR06] B. Adida and R. L. Rivest. Scratch & Vote: Self-contained Paper-based Cryptographic Voting. In *ACM WPES*, pages 29–40, 2006.
- [AR08] R. Araujo and P. Y. A. Ryan. Improving the Farnel Voting Scheme. In *EVOTE*, 2008.
- [Arr50] K. Arrow. A Difficulty in the Concept of Social Welfare. *Journal of Political Economy*, 58(4):328–346, 1950.
- [ARR⁺10] R. Araujo, N. B. Rajeb, R. Robbana, J. Traore, and S. Yousfi. Towards Practical and Secure Coercion-Resistant Electronic Elections. In *CANS*, 2010.
- [Bac02] A. Back. Hashcash: A Denial of Service Counter-measure, 2002.
- [BBTW12] R. Buckland, C. Burton, V. Teague, and R. Wen. The Future of E-voting in Australia. *IEEE Security and Privacy*, 99(Pre-print), 2012.
- [BCE⁺07] M. Blaze, A. Cordero, S. Engle, C. Karlof, N. Sastry, M. Sherr, T. Stegers, and K.-P. Yee. Source Code Review of the Sequoia Voting System. In *State of California’s Top to Bottom Review*, 2007.
- [BCJ⁺05] J. D. R. Buchanan, R. P. Cowburn, A.-V. Jausovec, D. Petit, P. Seem, G. Xiong, D. Atkinson, K. Fenton, D. A. Allwood, and M. T. Bryan. Fingerprinting Documents and Packaging. *Nature*, 436:475, 2005.
- [BDJR97] M. Bellare, A. Desai, E. Jokiphi, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption: An Analysis of the DES Modes of Operation. In *IEEE FOCS*, 1997.
- [BdM91] J. Benaloh and M. de Mare. Efficient Broadcast Time-Stamping. Technical Report TR-MCS-91-1, Clarkson University, 1991.
- [BdM93] J. Benaloh and M. de Mare. One-way Accumulators: A Decentralized Alternative to Digital Signatures. In *EUROCRYPT*, 1993.

- [BEH⁺08] K. Butler, W. Enck, H. Hursti, S. McLaughlin, P. Traynor, and P. McDaniel. Systemic Issues in the Hart InterCivic and Premier Voting Systems: Reflections on Project EVEREST. In *EVT*, 2008.
- [Ben86a] J. Benaloh. Cryptographic Capsules: A Disjunctive Primitive for Interactive Protocols. In *CRYPTO*, 1986.
- [Ben86b] J. Benaloh. Secret Sharing Homomorphisms: Keeping a Secret Secret. In *EUROCRYPT*, 1986.
- [Ben87] J. Benaloh. *Verifiable Secret-ballot Elections*. PhD thesis, Yale University, 1987.
- [Ben06] J. Benaloh. Simple Verifiable Elections. In *EVT*, 2006.
- [Ben07] J. Benaloh. Ballot Casting Assurance via Voter-Initiated Poll Station Auditing. In *EVT*, 2007.
- [Ben08] J. Benaloh. Administrative and Public Verifiability: Can We Have Both? In *EVT*, 2008.
- [BF85] J. D. Benaloh (*né* Cohen) and M. J. Fisher. A Robust and Verifiable Cryptographically Secure Election Scheme. In *SFCS*, 1985.
- [BGP11] P. Bulens, D. Giry, and O. Pereira. Running Mixnet-Based Elections with Helios. In *EVT/WOTE*, 2011.
- [BGR11] S. Bursuc, G. S. Grewal, and M. D. Ryan. Trivitas: Voters Directly Verifying Votes. In *VOTE-ID*, 2011.
- [BHK⁺09] J. M. Bohli, C. Henrich, C. Kempka, J. Mueller-Quade, and S. Röhrich. Enhancing Electronic Voting Machines on the Example of Bingo Voting. *IEEE Transactions on Information Forensics and Security*, 4:4:745–750, 2009.
- [BHMQ⁺08] M. Bär, C. Henrich, J. Mueller-Quade, S. Röhrich, and C. Stüber. Real World Experiences with Bingo Voting and a Comparison of Usability. In *WOTE*, 2008.
- [BHP⁺09] D. Bismark, J. Heather, R. Peel, S. Schneider, Z. Xia, and P. Ryan. Experiences Gained from the first Prêt à Voter Implementation. In *RE-VOTE*, pages 19–28, aug. 2009.
- [BHS91] D. Bayer, S. A. Haber, and W. S. Stornetta. Improving the Efficiency and Reliability of Digital Time-stamping. In *Sequences*, 1991.

- [BLLV98] A. Buldas, P. Laud, H. Lipmaa, and J. Villemson. Time-stamping with Binary Linking Schemes. In *CRYPTO*, 1998.
- [BMR07] J. M. Bohli, J. Mueller-Quade, and S. Roehrich. Bingo Voting: Secure and Coercion-free Voting Using a Trusted Random Number Generator. In *VOTE-ID*, pages 111–124, 2007.
- [BN00] D. Boneh and M. Naor. Timed Commitments. In *CRYPTO*, 2000.
- [Bor07] B. Borchert. Segment-based Visual Cryptography. Technical Report WSI-2007-04, WSI, 2007.
- [Bos92] J. Bos. *Practical Privacy*. PhD thesis, Technische Universiteit Eindhoven, 1992.
- [BR05] M. Bellare and P. Rogaway. Introduction to Modern Cryptography. In *UCSD CSE 207 Course Notes*, page 207, 2005.
- [BT94] J. Benaloh and D. Tuinstra. Receipt-free Secret-ballot Elections. In *ACM STOC*, 1994.
- [BY86] J. Benaloh and M. Yung. Distributing the Power of a Government to Enhance the Privacy of Voters. In *ACM PODC*, 1986.
- [CBH⁺11] C. Culnane, D. Bismark, J. Heather, S. Schneider, S. Srinivasan, and Z. Xia. Authentication Codes. In *EVT/WOTE*, 2011.
- [CCC⁺08] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman. Scantegrity II: End-to-end Verifiability for Optical Scan Election Systems Using Invisible Ink Confirmation Codes. In *EVT*, 2008.
- [CCC⁺09] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, A. T. Sherman, and P. L. Vora. Scantegrity II: End-to-end Verifiability by Voters of Optical Scan Elections Through Confirmation Codes. *IEEE Transactions on Information Forensics and Security*, 4:4(4):611–627, 2009.
- [CCC⁺10] R. T. Carback, D. Chaum, J. Clark, J. Conway, A. Essex, P. S. Hersonson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, A. T. Sherman, and P. L. Vora. Scantegrity II Municipal Election at Takoma Park: the first E2E Binding Governmental Election with Ballot Privacy. In *USENIX Security Symposium*, 2010.

- [CCEP07] R. T. Carback, J. Clark, A. Essex, and S. Popoveniuc. On the Independent Verification of a Punchscan Election. In *Proc. VoComp*, 2007.
- [CCF11] J. A. Calandrino, W. Clarkson, and E. W. Felten. Bubble Trouble: Off-Line De-Anonymization of Bubble Forms. In *USENIX Security Symposium*, 2011.
- [CCM08] M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a Secure Voting System. In *IEEE Symposium on Security and Privacy*, pages 354–368, 2008.
- [CD01] R. Cramer and I. Damgård. Multiparty Computation from Threshold Homomorphic Encryption. In *EUROCRYPT*, 2001.
- [CE12] J. Clark and A. Essex. CommitCoin: Commitments with Temporal Dispute Resolution using Bitcoin (Short Paper). In *FC*, 2012.
- [CEA07a] J. Clark, A. Essex, and C. Adams. On the Security of Ballot Receipts in E2E Voting Systems. In *WOTE*, 2007.
- [CEA07b] J. Clark, A. Essex, and C. Adams. Secure and Observable Auditing of Electronic Voting Systems Using Stock Indices. In *IEEE Canadian Conference on Electrical and Computer Engineering*, 2007.
- [CEC⁺08] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. T. Sherman, and P. Vora. Scantegrity: End-to-End Voter Verifiable Optical-Scan Voting. *IEEE Security and Privacy*, 6(3):40–46, May/June 2008.
- [CFH⁺07] J. A. Calandrino, A. J. Feldman, J. A. Halderman, D. Wagner, H. Yu, and W. P. Zeller. Source Code Review of the Diebold Voting System. In *State of California's Top to Bottom Review*, 2007. <http://www.cs.princeton.edu/~harlanyu/papers/dieboldsrc-ttbr.pdf>.
- [CFK⁺09] S. Checkoway, A. J. Feldman, B. Kantor, J. A. Halderman, E. W. Felten, and H. Shacham. Can DREs Provide Long-Lasting Security? The Case of Return-Oriented Programming and the AVC Advantage. In *EVT/WOTE*, 2009.
- [CFSY96] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-authority Secret-ballot Elections with Linear Work. In *EUROCRYPT*, 1996.
- [CGS97] R. Cramer, R. Gennaro, and B. Schoenmakers. A Secure and Optimally Efficient Multi-authority Election Scheme. In *EUROCRYPT*, 1997.
- [CH10] J. Clark and U. Hengartner. On the Use of Financial Data as a Random Beacon. In *EVT/WOTE*, 2010.

- [CH11] J. Clark and U. Hengartner. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. In *FC*, 2011.
- [Cha81] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
- [Cha88] D. Chaum. Elections with Unconditionally-Secure Ballots and Disruption Equivalent to Breaking RSA. In *EUROCRYPT*, 1988.
- [Cha04] D. Chaum. Secret-Ballot Receipts: True Voter-Verifiable Elections. *IEEE Security and Privacy*, 2(1):38–47, 2004.
- [CHL09] J. Clark, U. Hengartner, and K. Larson. Not-so-hidden Information: Optimal Contracts for Undue Influence in E2E Voting Systems. In *VOTE-ID*, 2009.
- [CKW08] J. Cichoń, M. Kutylowski, and B. Węglorz. Short Ballot Assumption and Threeballot Voting Protocol. In *SOFSEM*, 2008.
- [CLRS00] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd edition*. MIT Press, McGraw-Hill Book Company, 2000.
- [CMSW09] L. Chen, P. Morrissey, N. P. Smart, and B. Warinschi. Security Notions and Generic Constructions for Client Puzzles. In *ASIACRYPT*, 2009.
- [Con09] L. Coney. Report on the Manual Ballot Audit: Takoma Park, Maryland, November 3, 2009 Election, 2009. Electronic Privacy Information Center.
- [CP92] D. Chaum and T. P. Pedersen. Wallet Databases with Observers. In *CRYPTO*, 1992.
- [CPSC07] R. T. Carback, S. Popoveniuc, A. T. Sherman, and D. Chaum. Punchscan with Independent Ballot Sheets: Simplifying Ballot Printing and Distribution with Independently Selected Ballot Halves. In *WOTE*, 2007.
- [CRS05] D. Chaum, P. Y. A. Ryan, and S. Schneider. A Practical Voter-Verifiable Election Scheme. In *ESORICS*, 2005.
- [Cus01] R. F. Custodio. Farnel: Um Protocolo de Votacao Papel com Verificabilidade Parcial. In *Simposio Seguranca em Informatica*, 2001.
- [CWD06] A. Cordero, D. Wagner, and D. Dill. The Role of Dice in Election Audits. In *WOTE*, 2006.

- [CWF⁺09] W. Clarkson, T. Weyrich, A. Finkelstein, N. Heninger, J. A. Halderman, and E. W. Felten. Fingerprinting Blank Paper using Commodity Scanners. In *IEEE Symposium on Security and Privacy*, 2009.
- [Dil07] D. Dill. Election Vaporware. Huffington Post, September 2007. http://www.huffingtonpost.com/david-dill/election-vaporware_b_65810.html.
- [DMR06] S. Doshi, F. Monrose, and A. D. Rubin. Efficient Memory Bound Puzzles Using Pattern Databases. In *ACNS*, 2006.
- [DN92] C. Dwork and M. Naor. Pricing via Processing or Combatting Junk Mail. In *CRYPTO*, 1992.
- [DNW96] B. Davenport, A. Newberger, and J. Woodward. Creating a Secure Digital Voting Protocol for Campus Elections. Technical report, Princeton, 1996.
- [DS01] D. Dean and A. Subblefield. Using Client Puzzles to Protect TLS. In *USENIX Security*, 2001.
- [ECA08] A. Essex, J. Clark, and C. Adams. Aperio: High Integrity Elections for Developing Countries. In *WOTE*, 2008.
- [ECCP07a] A. Essex, J. Clark, R. T. Carback, and S. Popoveniuc. Punchscan in Practice: an E2E Election Case Study. In *WOTE*, 2007.
- [ECCP07b] A. Essex, J. Clark, R. T. Carback, and S. Popoveniuc. The Punchscan Voting System. Technical report, Submission to the *University Voting Systems Competition (VoComp)*, 2007.
- [ECHA09] A. Essex, J. Clark, U. Hengartner, and C. Adams. How to Print a Secret. In *HotSec*, 2009.
- [ECHA10] A. Essex, J. Clark, U. Hengartner, and C. Adams. Eperio: Mitigating Technical Complexity in Cryptographic Election Verification. In *EVT/WOTE*, 2010.
- [ECHA12] A. Essex, J. Clark, U. Hengartner, and C. Adams. Eperio: Mitigating Technical Complexity in Cryptographic Election Verification. In *IACR Eprint Report 2012/178*, 2012.
- [EH12a] A. Essex and U. Hengartner. HOVER: Trustworthy Elections with Hash-only Verification. *IEEE Security and Privacy*, To appear, 2012.
- [EH12b] A. Essex and U. Hengartner. Oblivious Printing of Secret Messages in a Multi-party Setting. In *FC*, 2012.

- [EHH11] A. Essex, C. Henrich, and U. Hengartner. Single layer optical-scan voting with fully distributed trust. In *VOTE-ID*, 2011.
- [Ess08] A. Essex. Punchscan: Designing an Independent Verification Mechanism for Elections. Master’s thesis, University of Ottawa, 2008.
- [FCS06] K. Fisher, R. T. Carback, and A. T. Sherman. Punchscan: Introduction and System Definition of a High-Integrity Election System. In *WOTE*, 2006.
- [Fed09] Federal constitutional court of Germany (Bundesverfassungsgericht). Judgement of 3 March 2009 – 2 BvC 3/07, 2 BvC 4/07 – Verfahren über die Wahlprüfungsbeschwerden, 2009.
- [FIP11] M. J. Fischer, M. Iorga, and R. Peralta. A Public Randomness Service. Technical report, National Institute of Standards and Technology, 2011.
- [FM97] M. K. Franklin and D. Malkhi. Auditable Metering with Lightweight Security. In *Financial Cryptography*, 1997.
- [FMM⁺02] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, and K. Sako. An Implementation of a Universally Verifiable Electronic Voting Scheme Based on Shuffling. In *FC*, 2002.
- [FMS10] J. Furukawa, K. Mori, and K. Sako. An Implementation of a Mix-Net Based Network Voting Scheme and Its Use in a Private Organization. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*. Springer, 2010.
- [FOO92] A. Fujioka, T. Okamoto, and K. Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *ASIACRYPT*, pages 244–251, 1992.
- [FS86] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO*, pages 186–194, 1986.
- [FSC09] R. A. Fink, A. T. Sherman, and R. Carback. TPM meets DRE: Reducing the Trust Base for Electronic Voting Using Trusted Platform Modules. *Trans. Info. For. Sec.*, 4(4):628–637, 2009.
- [Gee01] D. Geer. Technical Maturity, Reliability, Implicit Taxes, and Wealth Creation. *login: The magazine of Usenix & Sage*, 26(8), 2001.
- [GJMM98] E. Gabber, M. Jakobsson, Y. Matias, and A. Mayer. Curbing Junk E-mail Via Secure Classification. In *Financial Cryptography*, 1998.
- [GKR08] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. One-Time Programs. In *CRYPTO*, 2008.

- [GM84] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28:2:270-299, 1984.
- [GS98] D. M. Goldschlag and S. G. Stubblebine. Publicly Verifiable Lotteries: Applications of Delaying Functions. In *Financial Cryptography*, 1998.
- [Her97] M. A. Herschberg. Secure Electronic Voting Over the World Wide Web. Master's thesis, MIT, 1997.
- [HJP05] E. Hubbers, B. Jacobs, and W. Pieters. RIES: Internet Voting in Action. In *Computer Software and Applications Conference (COMPSAC)*, pages 417–424, 2005.
- [HJS⁺08] E. Hubbers, B. Jacobs, B. Schoenmakers, H. van Tilborg, and B. de Weger. Description and Analysis of the RIES Internet Voting System. Technical report, Eindhoven Institute for the Protection of Systems and Information (EiPSI), 2008.
- [HM96] S. Halevi and S. Micali. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing. In *CRYPTO*, 1996.
- [HRT10] J. Heather, P. Y. A. Ryan, and V. Teague. Pretty Good Democracy for More Expressive Voting Schemes. In *ESORICS*, 2010.
- [HS90] S. Haber and W. S. Stornetta. How to Time-Stamp a Digital Document. In *CRYPTO*, 1990.
- [HS00] M. Hirt and K. Sako. Efficient Receipt-Free Voting Based on Homomorphic Encryption. In *EUROCRYPT*, 2000.
- [HS11] R. Haenni and O. Spycher. Secure Internet Voting on Limited Devices with Anonymized DSA Public Keys. In *EVT/WOTE*, 2011.
- [HSS09] K. Henry, D. R. Stinson, and J. Sui. The Effectiveness of Receipt-Based Attacks on ThreeBallot. *IEEE TIFS*, 4(4):699–707, 2009.
- [IRS⁺07] S. Inguva, E. Rescorla, H. Shacham, , and D. S. Wallach. Source Code Review of the Hart InterCivic Voting System. In *State of California's Top to Bottom Review*, 2007.
- [JB99] A. Juels and J. Brainard. Client Puzzles: A Cryptographic Defense Against Connection Depletion Attacks. In *NDSS*, 1999.
- [JCJ05] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant Electronic Elections. In *ACM WPES*, 2005.

- [JJ99] M. Jakobsson and A. Juels. Proofs of Work and Bread Pudding Protocols. In *Communications and Multimedia Security*, 1999.
- [JJ00] M. Jakobsson and A. Juels. Mix and Match: Secure Function Evaluation via Ciphertexts. In *ASIACRYPT*, 2000.
- [JJB06] H. Jones, J. Juang, and G. Belote. ThreeBallot in the Field. MIT Course Project, 2006.
- [JJR02] M. Jakobsson, A. Juels, and R. L. Rivest. Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. In *USENIX Security Symposium*, pages 339–353, 2002.
- [Jon09] D. W. Jones. Some Problems with End-to-End Voting. In *NIST End-to-End Voting Systems Workshop*, 2009.
- [JR07] R. Joaquim and C. Ribeiro. CodeVoting: Protection Against Automatic Vote Manipulation in an Uncontrolled Environment. In *VOTE-ID*, 2007.
- [JR11] R. Joaquim and C. Ribeiro. An Efficient and Highly Sound Voter Verification Technique and its Implementation. In *VOTE-ID*, 2011.
- [JRF09] R. Joaquim, C. Ribeiro, and P. Ferreira. VeryVote: A Voter Verifiable Code Voting System. In *VOTE-ID*, 2009.
- [JRF10] R. Joaquim, C. Ribeiro, and P. Ferreira. Improving Remote Voting Security with CodeVoting. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*. Springer, 2010.
- [KC10] G. O. Karame and S. Capkun. Low-cost Client Puzzles Based on Modular Exponentiation. In *ESORICS*, 2010.
- [KHF11] R. Koenig, R. Haenni, and S. Fischli. Preventing Board Flooding Attacks in Coercion-Resistant Electronic Voting Schemes. In *SEC*, 2011.
- [KK87] O. Kafri and E. Keren. Encryption of Pictures and Shapes by Random Grids. *Optics Letters*, 12:6:377–379, 1987.
- [KRMC07] J. Kelsey, A. Regenscheid, T. Moran, and D. Chaum. Hacking Paper: Some Random Attacks on Paper-Based E2E Systems. In *Frontiers of Electronic Voting*, 2007.
- [KRMC10] J. Kelsey, A. Regenscheid, T. Moran, and D. Chaum. Attacking Paper-Based E2E Voting Systems. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*, pages 370–387. Springer, 2010.

- [KSRW04] T. Kohno, A. Stubblefield, A. D. Rubin, and D. S. Wallach. Analysis of an Electronic Voting System. In *IEEE Symposium on Security and Privacy*, 2004.
- [KTV09] R. Küsters, T. Truderung, and A. Vogt. Improving and Simplifying a Variant of Prêt à Voter. In *VOTE-ID*, 2009.
- [KTV10] R. Küsters, T. Truderung, and A. Vogt. Proving Coercion-Resistance of Scantegrity II. In *ICICS*, 2010.
- [Kub06] P. Kubiak. A Modification of Punchscan: Trust Distribution. In *FEE*, 2006.
- [KZ07] M. Kutylowski and F. Zagorski. Verifiable Internet Voting Solving Secure Platform Problem. In *IWSEC*, 2007.
- [KZ10] M. Kutylowski and F. Zagorski. Scratch, Click & Vote: E2E Voting Over the Internet. In *Towards Trustworthy Elections*. Spr, 2010.
- [LNS⁺02] S.-S. Lee, J.-C. Na, S.-W. Sohn, C. Park, D.-H. Seo, and S.-J. Kim. Visual Cryptography Based on an Interferometric Encryption Technique. *ETRI*, 24(5):373–380, 2002.
- [LR08] D. Lundin and P. Y. Ryan. Human Readable Paper Verification of Prêt à Voter. In *ESORICS*, 2008.
- [LTR⁺06] D. Lundin, H. Treharne, P. Y. A. Ryan, S. Schneider, J. Heather, and Z. Xia. Tear and Destroy: Chain Voting and Destruction Problems Shared by Prêt à Voter and Punchscan and a Solution Using Visual Encryption. In *FEE*, 2006.
- [LWL09] F. Liu, C. K. Wu, and X. J. Lin. The Alignment Problem of Visual Cryptography Schemes. *Designs, Codes and Cryptography*, 50(2), 2009.
- [MB02] P. Maniatis and M. Baker. Enabling the Long-Term Archival of Signed Documents through Time Stamping. In *FAST*, 2002.
- [MMV11] M. Mahmoody, T. Moran, and S. Vadhan. Time-lock Puzzles in the Random Oracle Model. In *CRYPTO*, 2011.
- [MN05] T. Moran and M. Naor. Basing Cryptographic Protocols on Tamper-Evident Seals. In *In Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, pages 285–297, 2005.
- [MN07] T. Moran and M. Naor. Split-Ballot Voting: Everlasting Privacy With Distributed Trust. In *ACM CCS*, 2007.

- [MSTS04] T. Moran, R. Shaltiel, and A. Ta-Shma. Non-interactive Timestamping in the Bounded Storage Model. In *CRYPTO*, 2004.
- [MVM11] M. Mahmoody, S. P. Vadhan, and T. Moran. Non-Interactive Time-Stamping and Proofs of Work in the Random Oracle Model. IACR ePrint 553, 2011.
- [Nak08] S. Nakamoto. Bitcoin: A Peer-to-peer Electronic Cash System. Unpublished, 2008. <http://www.bitcoin.org/bitcoin.pdf>.
- [Nef04] C. A. Nef. Practical High Certainty Intent Verification for Encrypted Votes. Technical report, VoteHere Whitepaper, 2004.
- [NS94] M. Naor and A. Shamir. Visual Cryptography. In *EUROCRYPT*, 94.
- [Pai99] P. Paillier. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*, 1999.
- [PC10] S. Popoveniuc and R. Carback. ClearVote: An End-to-End Voting System that Distributes Privacy Between Printers. In *WPES*, 2010.
- [Ped91] T. P. Pedersen. A Threshold Cryptosystem Without a Trusted Party. In *EUROCRYPT*, 1991.
- [Ped92] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO*, pages 129–140, 1992.
- [PERW03] N. Paul, D. Evans, A. D. Rubin, and D. S. Wallach. Authentication for Remote Voting. In *HCISS*, 2003.
- [PH06] S. Popoveniuc and B. Hosp. An Introduction to Punchscan. In *WOTE*, 2006.
- [PIK93] C. Park, K. Itoh, and K. Kurosawa. Efficient Anonymous Channel and All/nothing Election Scheme. In *EUROCRYPT*, 1993.
- [PKRV10] S. Popoveniuc, J. Kelsey, A. Regenscheid, and P. Vora. Performance Requirements for End-to-End Verifiable Elections. In *EVT/WOTE*, 2010.
- [PRQ⁺98] B. Preneel, B. V. Rompay, J. J. Quisquater, H. Massias, and J. S. Avila. Design of a Timestamping System. Technical Report WP3, TIMESEC Project, 1998.
- [PS07] S. Popoveniuc and J. Stanton. Undervote and Pattern Voting: Vulnerability and a Mitigation Technique. In *WOTE*, 2007.

- [PS08] S. Popoveniuc and J. Stanton. Buying Random Votes is as Hard as Buying No-votes. Technical report, ePrint Arichive, 2008.
- [PV08] S. Popoveniuc and P. L. Vora. A Framework for Secure Electronic Voting. In *WOTE*, 2008.
- [Rab83] M. Rabin. Transaction Protection by Beacons. *Journal of Computer and System Sciences*, 27(2), 1983.
- [RBH⁺09] P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia. Prêt à Voter: a Voter-Verifiable Voting System. *IEEE TIFS*, 4(4), 2009.
- [Row08] L. Rowell. Down for the Count. *ACM Networker*, 12:1:16–23, 2008.
- [RS96] R. L. Rivest and A. Shamir. PayWord and MicroMint: Two Simple Micro-payment Schemes. In *Security Protocols*, 1996.
- [RS06] P. Y. A. Ryan and S. A. Schneider. Prêt à Voter with Re-encryption Mixes. In *ESORICS*, 2006.
- [RS07] R. L. Rivest and W. D. Smith. Three Voting Protocols: ThreeBallot, VAV, and Twin. In *EVT*, 2007.
- [RSW96] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock Puzzles and Timed-release Crypto. Technical Report TR-684, MIT, 1996.
- [RT09a] P. Y. A. Ryan and V. Teague. Ballot Permutations in Prêt à Voter. In *EVT/WOTE*, 2009.
- [RT09b] P. Y. A. Ryan and V. Teague. Pretty Good Democracy. In *Workshop on Security Protocols*, 2009.
- [RWB⁺05] A. D. Rubin, D. S. Wallach, D. Boneh, M. D. Byrne, D. Dean, D. L. Dill, D. W. Jones, P. G. Neumann, D. Mulligan, and D. A. Wagner. A Center for Correct, Usable, Reliable, Auditable, and Transparent Elections (ACCU-RATE). NSF CyberTrust Research Proposal, 2005.
- [Rya11] P. Y. A. Ryan. Prêt à Voter with Confirmation Codes. In *EVT/WOTE*, 2011.
- [SCC⁺10] A. T. Sherman, R. T. Carback, D. Chaum, J. Clark, A. Essex, P. S. Herson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, B. Sinha, and P. L. Vora. Scantegrity Mock Election at Takoma Park. In *EVOTE*, 2010.

- [SDW08] D. R. Sandler, K. Derr, and D. S. Wallach. VoteBox: a Tamper-evident, Verifiable Electronic Voting System. In *USENIX Security Symposium*, 2008.
- [Sha79] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, November 1979.
- [SHD10] O. Spycher, R. Haenni, and E. Dubuis. Coercion-Resistant Hybrid Voting Systems. In *EVOTE*, 2010.
- [SHKS11] M. Schläpfer, R. Haenni, R. Koenig, and O. Spycher. Efficient Vote Authorization in Coercion-Resistant Internet Voting. In *VOTE-ID*, 2011.
- [SK94] K. Sako and J. Kilian. Secure Voting Using Partially Compatible Homomorphisms. In *CRYPTO*, 1994.
- [SK95] K. Sako and J. Kilian. Receipt-Free Mix-Type Voting Scheme - A Practical Solution to the Implementation of a Voting Booth. In *EUROCRYPT*, pages 393–403, 1995.
- [SKHS11] O. Spycher, R. Koenig, R. Haenni, and M. Schläpfer. A New Approach Towards Coercion-Resistant Remote E-Voting in Linear Time. In *FC*, 2011.
- [SKR⁺11] D. Stebila, L. Kuppusamy, J. Rangasamy, C. Boyd, and J. M. Gonzalez Nieto. Stronger Difficulty Notions for Client Puzzles and Denial-of-Service-Resistant Protocols. In *CT-RSA*, 2011.
- [Smi05] W. D. Smith. New Cryptographic Election Protocol with Best-known Theoretical Properties. In *Frontiers in Electronic Elections*, 2005.
- [SSB11] A. Sharma, L. Subramanian, and E. Brewer. PaperSpeckle: Microscopic Fingerprinting of Paper. In *CCS*, 2011.
- [TBFG07] S. Tritilanunt, C. Boyd, E. Foo, and J. M. Gonzalez Nieto. Toward Non-parallelizable Client Puzzles. In *CANS*, 2007.
- [THvLT05] P. Tuyls, H. D. L. Hollmann, J. H. v. Lint, and L. Tolhuizen. Xor-based Visual Cryptography Schemes. *Designs Codes and Cryptography*, 37:169–186, 2005.
- [Tze04] W. G. Tzeng. Efficient 1-Out-of-n Oblivious Transfer Schemes with Universally Usable Parameters. *IEEE Transactions on Computers*, 53(2), 2004.
- [Uni05] United States Election Assistance Commission. 2005 Voluntary Voting System Guidelines, v1, 2005.

- [Uni08] United States Election Assistance Commission. 2008 Election Administration & Voting Survey Report, 2008.
- [VK04] D. Q. Viet and K. Kurosawa. Almost Ideal Contrast Visual Cryptography with Reversing. In *CT-RSA*, volume 2964, pages 353–365, 2004.
- [WAB07] S. G. Weber, R. S. d. Araujo, and J. Buchmann. On Coercion-Resistant Electronic Elections with Linear Work. In *ARES*, 2007.
- [Wei08] T. R. Weiss. Q&A: E-voting Activist More Optimistic About Voting Systems. Computerworld, July 2008.
- [WJHF04a] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New Client Puzzle Outsourcing Techniques for DoS Resistance. In *ACM CCS*, pages 246–256, 2004.
- [WJHF04b] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New Client Puzzle Outsourcing Techniques for DoS Resistance. In *CCS*, 2004.
- [WR03] X. Wang and M. K. Reiter. Defending Against Denial-of-Service Attacks with Puzzle Auctions. In *IEEE Symposium on Security and Privacy*, 2003.
- [Wu05] H. Wu. The Misuse of RC4 in Microsoft Word and Excel. Cryptology ePrint Archive, Report 2005/007, 2005.
- [XSH08] Z. Xia, S. A. Schneider, and J. Heather. Analysis, Improvement and Simplification of Prêt à Voter with Paillier Encryption. In *EVT*, 2008.
- [Yan11] C.-N. Yang, editor. *Visual Cryptography and Secret Image Sharing*. CRC Press, 2011.

Appendices

Appendix A

Eperio Implementation Details

A.1 Eperio Audit Data Specification

All (unencrypted and decrypted) election data files are formatted as UTF-8 encoded Comma Separated Values (CSV). Columns are separated by the delimiter ‘,’ (0x2C) and rows are separated by a carriage return (0x0A).

File Naming and Formatting Convention. For an election of b ballots, c candidates, I proof instances, of which p were print audited, the following files will be generated and posted to the bulletin board through the course of the election verification timeline

`$electionDir$/AssertedReceipts.csv`: is a $bc \times 2$ list containing an assertion of all receipt information created during an election. Each element in $(:,1)$ is unique election references: $s-p$ where $1 \leq s \leq b$ is a ballot serial number and $0 \leq p \leq c - 1$ a mark position. Elements in column $(:,2)$ represent the corresponding mark state registered during the election, either 0, 1, or -1 indicating the corresponding unique reference was either unmarked, marked or print audited respectively.

`$electionDir$/AssertedTally.csv`: is a $bc \times 2$ list containing an assertion of aggregate voting intent created during an election. Column $(:,1)$ is a list of b repetitions of c candidate names. Elements in column $(:,2)$ represent the corresponding mark state.

`$electionDir$/LinkageList.csv`: is a $p \times i + 2$ list containing an assertion of the association between receipt $(:,1)$ and candidate $(:,2)$ for all print

audited ballots, as well as an assertion of the row number the entry in each of the I instances of shuffled receipt (U) and tally (S) lists.

`$electionDir$/ChallengeKeys.csv`: is a $i \times 2$ list containing a respond to the audit challenge. Elements at $(i, 1)$ represent the relative file path to the file that was challenged: the i -th shuffled instance of the receipt list `$electionDir$/i/Ui` or tally list `$electionDir$/i/Si` respectively. Elements at $(i, 2)$ represent the corresponding revealed decryption key to the file (128-bits, BASE64 encoded).

For each instance $1 \leq i \leq I$:

`$electionDir$/i/Si.csv`: is a $bc \times 3$ list representing a shuffled instance of column $(:, 1)$ of `AssertedTally.csv` repeated three times (as per the commitment scheme described in section 7.4) in columns $(:, 1)$, $(:, 2)$ and $(:, 3)$ respectively.

`$electionDir$/i/Ui.csv`: is a $bc \times 3$ list representing a shuffled instance of column $(:, 1)$ of `AssertedReceipts.csv` repeated three times as above.

`$electionDir$/Marks/Mi.csv`: is a $bc \times 1$ list representing a shuffled instance of column $(:, 2)$ of `AssertedReceipts.csv` and `AssertedTally.csv`.

File Encryption Conventions. 128-bit file encryption keys are derived by the semi-trusted trustee workstation and BASE64 encoded for convenience of transmission over the internet. The two file encryption options (OpenSSL and TrueCrypt) use their own password-based key derivation functions. OpenSSL and TrueCrypt encrypted files are denoted by the `.enc` and `.tcr` extensions, and use AES-128-CBC and AES-128-XTS encryption modes respectively. There is no inherent way to bypass the key-derivation process of these applications to directly apply a binary key. As such the BASE64 encoded keys are supplied as passwords and each program (OpenSSL, TrueCrypt, etc) will derive its own (binary) encryption key using its own particular password based key derivation function. In OpenSSL the workstation applies the `-a` command option to BASE64 encode the encrypted output for safe internet transport.

A.2 Python Code for Eperio Verification

```
1 import csv, os, string, sys
2 def openCSV(fileName):
3     data = []
4     fileContents = open(fileName, 'r')
5     for i in fileContents:
6         data.append(i.strip().split(","))
7     return data
8 def verifyFailed(errorMsg):
9     print errorMsg
10    sys.exit()
11 def verifyCommitment(instance):
12    os.system("openssl enc -d -aes-128-cbc -a -in " + instance[0] + ".enc -out " +
13              instance[0] + ".csv -pass pass:" + instance[1])
14    data = openCSV(instance[0].strip("'") + ".csv")
15    col = []
16    for i in data:
17        if i[0] != i[1] or i[0] != i[2]:
18            verifyFailed("Invalid Commitment")
19    return [i[0] for i in data]
20 def getMarks(instance):
21    marks = openCSV("./Marks/M"+ instance[0][2:4] + ".csv")
22    return [i[0] for i in marks]
23 assReceipts = sorted(openCSV('./AssertedReceipts.csv'))
24 assTally = sorted(openCSV('./AssertedTally.csv'))
25 pwds = openCSV('./ChallengeKeys.csv')
26 linkageList = openCSV('./LinkageList.csv')
27 for i, instance in enumerate(pwds):
28    merged=[]
29    data = verifyCommitment(instance)
30    marks = getMarks(instance)
31    for index in range(len(data)):
32        merged.append([data[index], marks[index]])
33    if instance[0][5] == "U" :
34        for ballot, linkage in enumerate(linkageList):
35            if linkageList[ballot][0] != merged[int(linkageList[ballot][i
36              +2]) - 1][0] or merged[int(linkageList[ballot][i+2]) - 1][1] !=
37              "-1" :
38                verifyFailed("Linkage_list_FAIL: Receipt_instance_#%d_
39                  does_not_does_not_match_linkage_list" % (i+1))
40            if assReceipts == sorted(merged):
41                print "Instance_%d_verified" % (i+1)
42            else:
43                verifyFailed("Verification_FAIL: Receipt_instance_#%d_does_not_
44                  match_asserted_receipt_list!" % (i+1))
45        elif instance[0][5] == "S":
46            for ballot, linkage in enumerate(linkageList):
47                if linkageList[ballot][1] != merged[int(linkageList[ballot][i
48                  +2]) - 1][0] or merged[int(linkageList[ballot][i+2]) - 1][1] !=
49                  "-1" :
50                    verifyFailed("Linkage_list_FAIL: Tally_instance_#%d_do\
51                      texttt{BASE64}_encoded_does_not_does_not_match_linkage_
52                      list" % (i+1))
53                if assTally == sorted(merged):
54                    print "Instance_%d_verified" % (i+1)
55                else:
```

```

48         verifyFailed(" Verification_FAIL: Tally_instance_#%s_does_not_
49         match_asserted_tally_list!" % (i+1))
50     else:
        verifyFailed(" Instance_filename_is_incorrectly_formatted")

```

A.3 User Instructions for Manual/Spreadsheet Verification

The following instructions were tested in OpenOffice.org Calc 3.0 and TrueCrypt 6.0a. The user is assumed to have downloaded all relevant data files to directory `/$selectionDir$/. Let i represent the instance number the verifier wishes to check.`

Part I: Verify Commitments

In Calc:

- Open `$selectionDir$/ChallengeKeys.csv`
- Let `$filePath$` represent the contents of cell A_i

In TrueCrypt:

- Select a free drive letter (without loss of generality assume Z:)
- Click `Select File` → `Select $selectionDir$/$filePath$.trc` → `Open`

In Calc:

- Click on cell B_i → `CTRL + C`

In TrueCrypt:

- Click `Mount` → `CTRL+V` → `OK`

In Calc:

- Open `Z:/$filePath$.csv`,
- In cell D1 type formula `=A1=B1`,

- Click on cell D1 once and double click on the square in the bottom right corner of the cell to apply the formula to all cells in the column,
- In cell E1 type formula =A1=C1,
- Click on cell E1 once and double click on the square as before,
- In cell F1 type formula =COUNTIF(D1:E65536;0),¹
- If F1 equals 0, the commitment is verified.

Part II: Comparing decommitted list to asserted list

In Calc:

- Let `$asserted$` represent `$electionDir$/AssertedReceipts.csv` if `$filePath$` is `/i/Ui`, otherwise let it represent `$electionDir$/AssertedTally.csv` if `$filePath$` is `/i/Si`
- Open `$electionDir$/ $asserted$.csv`,
- Click Data → Sort. Select Sort by Column A (Ascending) and Then by Column B (Ascending) → OK,
- In `Z:/$filePath$.csv` click on column A → CTRL+C,
- Open `$electionDir$/Marks/Mi.csv`,
- Right click on column A → Insert Columns → CTRL+V,
- Click Data → Sort. Select Sort by Column A (Ascending) and Then by Column B (Ascending) → OK,
- Holding CTRL click columns A and B → CTRL+C,
- In the asserted list, click column C → CTRL+V,
- In cell E1 type formula =A1=C1 and apply formula to all cells in the column as previously described,
- In cell F1 type formula =B1=D1 and apply to all cells in the column,
- In cell G1 type formula =COUNTIF(E1:F65536;0),

¹65536 is the maximum row number in OpenOffice 3.0.

- If G1 equals 0, Z:/\$filePath\$.csv matches \$electionDir\$/AssertedReceipts.csv.

PART III: Comparing decommitted list to linkage list

In Calc:

- Let $X = A$ if the asserted receipt list was opened during Part II otherwise let $X = B$ if the asserted tally was opened instead. Let Y be the $i + 2$ -th letter of the alphabet,
- Open \$electionDir\$/LinkageList.csv,
- For each print audited ballot $1 \leq j \leq p$:
 - Click on cell $Yj \rightarrow \text{CTRL+C}$,
 - In Z:/\$filePath\$.csv in the name box (top left) type ‘A’ $\rightarrow \text{CTRL+V} \rightarrow \text{Enter}$. The audit fails if the contents of the active (highlighted) cell do not match the contents of cell Xj in the linkage list.
 - In \$electionDir\$/Marks/Mi.csv in the name box type ‘A’ $\rightarrow \text{CTRL+V} \rightarrow \text{Enter}$. The audit fails if the contents of the active cell does not equal -1.

Appendix B

Carbon Dating the 2011 Takoma Park Pre-election Commitments

The Scantegrity pre-election commitments were made with CommitCoin on Oct 18, 2011 for the municipal election of Takoma Park, MD held on Nov 08, 2011. The 6 MeetingOneOut.xml files from the Scantegrity data (which contain the pre-election commitments of the 6 wards of Takoma Park's election) were inserted into the block chain using the same simplified version of CommitCoin used for the proof-of-concept above. Since the files already contained randomized commitments generated by Scantegrity, we simply hashed them to an appropriate size. The Bitcoin blockchain will show BTC0.01 was sent to the hash of each of these 6 files.

Bitcoin's blockchain forms a proof-of-work. Participants in the Bitcoin network use their computers to compete to "solve blocks" (*i.e.*, to find partial hash preimages). The average number of hashes required to solve a block at the current difficulty level is 2^{52} . The network is currently able to solve one block on average every 12 minutes. An adversary attempting to change the commitments of the Takoma Park election (e.g., on election night, Nov 8th, 2011) would have to produce an alternate (but valid) block chain, which would require them to compute over 2^{63} hashes. As the block chain grows through time (through the course of commerce done with Bitcoin), so would the attacker's work load.

We used the following approach:

1. Convert file to hash: $\text{RIPEMD160}(\text{file}) = \text{hash}$
2. Convert hash to Bitcoin address format: $\text{Hash2Address}(\text{hash}) = \text{BitcoinAddress}$
3. Send funds to BitcoinAddress: URL of transaction

The files already contain the commitments. The transactions below appeared in the Bitcoin blockchain at (2011-10-18 17:26:00):

```
baseURL = https://scantegrity.org/svn/data/takoma-nov8-2011/

RIPEMD160(ward1/MeetingOneOut.xml) = f6458eceedf326af9d4fe74125bdc2e762d28ac9
http://blockexplorer.com/q/hashtoaddress/f6458eceedf326af9d4fe74125bdc2e762d28ac9 =
    1PTAZ9wMZ2Ff9RgLt4UMXMh7vbBNdTDVbs
Transaction:
http://blockexplorer.com/tx/3789397fc352e93e7f1e7be3b770a04bff251ae36fa601125372336c626cb743

RIPEMD160(ward2/MeetingOneOut.xml) = d2a8535ba5a61bad576d2adecb54c700c40ae2d4
http://blockexplorer.com/q/hashtoaddress/d2a8535ba5a61bad576d2adecb54c700c40ae2d4 =
    1LCrYkh7nDRippJXx79kzVRvsbG13gKagN
Transaction:
http://blockexplorer.com/tx/769104a1676b56232a1e64f3001c874926945ff1ca2854484ccea32faf10621a

RIPEMD160(ward3/MeetingOneOut.xml) = abc960bcd48b89b8b2a8e6fdb3713d6f2a50ecf5
http://blockexplorer.com/q/hashtoaddress/abc960bcd48b89b8b2a8e6fdb3713d6f2a50ecf5 =
    1GfKnsSffFBETb3334M19LX5GCTqjNV3Du
Transaction:
http://blockexplorer.com/tx/9a7c41157b09a5df78cbeb0b9158d310639eded63a015fe571ba96c8dd012903

RIPEMD160(ward4/MeetingOneOut.xml) = a6866ea967e326fe9f28f8ae76ea32a396cb5f29
http://blockexplorer.com/q/hashtoaddress/a6866ea967e326fe9f28f8ae76ea32a396cb5f29 =
    1GBWDMzPjREp1kh6UDKfrqzJySpMnKJCF8
Transaction:
http://blockexplorer.com/tx/a265acda0eb4f71fd6655894810d11b268f1e4de56dd018a90897f7d6c28a4ce

RIPEMD160(ward5/MeetingOneOut.xml) = 5dce8714c84d7df569e4c4dc7dad24fd3d8aeccc
http://blockexplorer.com/q/hashtoaddress/5dce8714c84d7df569e4c4dc7dad24fd3d8aeccc =
    19Z1FdkgY4ab7S2oF2mrVJmHTkbRNsZV3X
Transaction:
http://blockexplorer.com/tx/630630365bd80cf0c5011709bb23f2f3bd4e9944c513ba79d2d43b45fdb0e848

RIPEMD160(ward6/MeetingOneOut.xml) = 8e62d49a002b35e5463c887a8961739d70d45ac5
http://blockexplorer.com/q/hashtoaddress/8e62d49a002b35e5463c887a8961739d70d45ac5 =
    1DysM6u7Mu7Gfp1cug6EpBSshijZg1CakY
Transaction:
http://blockexplorer.com/tx/0a1f9361f068c1f3f05f16a8bea89173ad20045bd6f5e6f57611ce6e925185f5
```