Performance Optimization of a Split-Value Voting System

by

Charles Z. Liu

Submitted to the
Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

Author .....
Department of Electrical Engineering and Computer Science
June 8, 2015

Certified by .....
Prof. Ronald L. Rivest
Thesis Supervisor

Accepted by .....
Prof. Albert R. Meyer
Chairman, Masters of Engineering Thesis Committee

Performance Optimization of a Split-Value Voting System

by

Charles Z. Liu

Submitted to the Department of Electrical Engineering and Computer Science
on June 8, 2015, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

TODO: ABSTRACT

Thesis Supervisor: Prof. Ronald L. Rivest

Acknowledgments

TODO: ACKNOWLEDGEMENTS

Contents

Chapter 1

Introduction

As a cornerstone of democratic governments, elections are the subject of intense scrutiny. Voters expect that the systems and procedures put in place are easy to follow, efficient, and inexpensive. However, history tells us that in practice, this may not always be the case.

Perhaps the most well-known example of the failure of voting systems in the current generation is the 2000 Presidential Election in the United States, most notably the series of recounts in Florida that delayed the nationwide election result by several weeks and opened cases in the Florida Supreme Court and the United States Supreme Court. Even in the aftermath of the election, studies were inconclusive as to whether the county-specific recounts or a statewide recount would have swung the results in Al Gore's favor. The problem was exacerbated by the media's premature declaration of the election outcome, George W. Bush's narrow margin of victory (around 500 votes), and controversy surrounding the "butterfly ballots" that were used in Palm Beach for the first time.

Studies conducted after the 2000 Election in Florida reveal many issues that appear to be quite subtle in the larger picture of a Presidential election: the public's familiarity with a new ballot layout and instructions, the timing of media releases and exit polls, and even the degree to which paper ballots are cleanly marked or punctured by the voter. In the aftermath of the chaos, Congress passed the Help America Vote Act, which attempted to help states upgrade their election infrastructure to pre-

vent similar issues four years later. Many states turned to electronic voting systems. Unfortunately, these systems have been met with criticisms over their accessibility, relability, and security. Furthermore, many of these proprietary systems have reached or are approaching the ends of their lifetimes, prompting another wave of research and development on secure, verifiable, efficient, and reliable electronic voting systems.

Elections are not only thwarted by the failures of the mechanisms used in them. Reports of manipulated or lost ballots, coerced voters, and simply corrupt election officials are no surprise in certain parts of the world even today. In decades past, voters in Italian elections often had the option of selecting ranked preferences among a long list of candidates. The clever "Italian attack" reportedly used by the Mafia involved coercing a voter to vote for the coercer's first choice candidate, followed by a sequence of low-probability candidates in some predetermined order. With a long list of candidates, the chance that another ballot contains the exact same sequence in the same slots is very low; thus, if the votes are publicly broadcast following the election, the sequence becomes a signature for the coerced voter that the coercer can check, even if all other identifiable information is stripped.

Clearly, there is much more to be done in advancing election systems. In this section, we briefly survey the history of voting systems, including electronic voting systems, and conclude with a discussion of desirable properties in electronic voting systems.

1.1 A Brief History of Voting Systems

The first recorded elections occurred in the sixth century BC in ancient Greece, where adult males charged with the responsibility to vote at meetings of the assembly. In the beginning, voting was typically done by a raise of hands in a public space. This developed into the use of colored stones, where voters would select a colored stone out of two possible colors and place it into a central jar. Although we use secret ballots today without a second thought, the first "secret ballots" were used in the ancient Greek and Roman civilizations, where records exist of contraptions that

allowed a voter to hide the stone as it was deposited into the jar. By the end of the nineteenth century, most Western countries, including the United States, Australia, and France, demanded the use of a secret ballot at elections, and voter privacy was a non-negotiable property of official elections.

As populations grew and election officials looked for ways to make elections more efficient, mechanized voting machines were developed. Lever machines were introduced in New York during the 1892 Election. Punch card systems were introduced in 1965 and were also widely used until they drew sharp criticism during the 2000 Election in Florida. Lever machines, punch cards, and other mechanical and electrical systems such as optical scanners provided different tradeoffs in cost, convenience, efficiency, usability, accessibility, and security. However, in the past decade, growth in both the power and popularity of digital technology has enabled a shift towards electronic voting systems.

1.2 The Rise of Electronic Voting

In the aftermath of the 2000 Election, the Help America Vote Act instituted a series of requirements on the states which aimed to replace punchcard and lever machines, establish standards on voter privacy and accessibility for voters with disabilities, and improve the usability of voting systems. The number of direct-recording electronic (DRE) voting machines grew in response. DREs have a number of advantages over their mechanical counterparts.

- They enable rapid, accurate tallying of votes.
- They may include numerous accessibility functions, notably those for the visually impaired.
- They may include many languages.
- They may catch errors such as undervoting and overvoting before the vote is cast, preventing invalid ballots from being discarded.

However, DREs are often complex systems containing proprietary hardware and software, making it difficult to verify their correctness and security properties; indeed, studies have shown the existence of numerous security vulnerabilities in many commercially available electronic voting systems. Because of the lack of a paper trail, any errors in the system are unrecoverable.

A solution to the lack of verifiability in DREs was proposed in 1992 by Mercuri [CITE THIS], creating a voter-verified paper audit trail (VVPAT) system. In the simplest VVPAT system, a DRE machine is augmented with a printer. In addition to submitting the vote electronically, the machine prints out a human-readable copy of the voter's selections. The voter may then confirm that the paper record is indeed correct before depositing it into a ballot box. In case of a disputed election, recount, or malfunctioning machines, the paper records may be consuled. VVPAT systems may be extended so that the voters may keep a printed receipt; however, care must be taken to ensure that the printed receipt does not reveal the voter's vote to anyone else.

Today, VVPAT systems are the most common type of electronic voting system; 27 states in the United States require a paper audit trail by law and an additional 18 states use them in their elections.

1.3 Desirable Properties in Electronic Voting Systems

The two main propeties desired in an election are *privacy* and *verifiability*.

Privacy concerns the ability of an adversary to gain information on a voter's choices in an election. A lack of privacy results in the possibility of coercion or bribery as seen in the earlier example of the "Italian attack". Note that the adversary is not limited to an outside third party; privacy is broken if a corrupt election official may associate any vote with the corresponding voter. As voting systems have become more complex and electronic voting systems have become more popular, the notion

of privacy has evolved.

The simplest form of privacy is *ballot privacy*, which stipulates that the contents of the ballot must remain secret. With typical paper ballots such as those used today in the United States, ballot privacy guarantees that there can be no association made between a voter and her ballot, thereby providing privacy to the voter. However, with electronic voting systems, a stronger definition of privacy is desired. *Receipt freeness* or *incoercibility*, first introduced in [BT94], concerns the inability of the voter to later prove how she voted. Equivalently, it is impossible to effectively coerce a voter by forcing her to reveal her vote following the election. The concept of receipt freeness grew in importance as electronic voting systems often leave a receipt that the voter may bring out of the polling site.

In certain areas of the United States and other countries including Switzerland and Estonia, Internet voting has become commonplace. In Estonia, for example, voters may scan their government issued ID cards to authenticate themselves into a voting website. Many more states and countries have vote-by-mail, in which ballots may be mailed ahead of the election date. A stronger notion of *coercion resistance* due to the work of [JCJ05] identified the need to prevent adversaries from jeopardizing voter privacy in these cases, for example, by stealing or forcing a voter to reveal her authentication credentials.

Verifiability concerns whether individual voters or the public as a whole may verify that the election outcome faithfully represents the selections of the voters. In particular, *individual verifiability* refers to the ability of a voter to verify that her vote was recorded in the election. *Universal verifiability* refers to the ability of anyone in the public to verify that the declared election outcome matches the set of votes that were recorded in the election.

Although today's paper ballots provide good privacy, they do so at immense cost to verifiability. Despite thorough documentation of best practices concerning how ballots should be handled, who should touch them, how they should be sealed, etc. [ACE Electoral Network], it remains practically impossible for a voter to know for certain that her ballot was not lost in transit. Since ballots are not released to the

public, voters must trust that the outcome declared by election officials represents the accurate tallying of all ballots. As seen in the 2000 Presidential Election example, good verifiability is not a strength in the systems used in the majority of the United States.

For electronic voting systems, however, we may achieve verifiability while preserving privacy by using cryptography where appropriate. For example, a vote may be encrypted before being cast; the encrypted votes may then be added homomorphically to yield an encryption of the final election outcome, enabling the election officials to compute the result of an election without revealing any individual votes. Threshold encryption may strengthen this. For example, by requiring that 3 out of 5 election officials be present to obtain the decryption key, we reduce the chance of a malicious election official revealing individual votes.

By introducing a secure, publicly viewable bulletin board, we may achieve verifiability at every stage from when the vote is cast until the election result is revealed. We define an *end-to-end verifiable* voting system as one that satisfies the following properties.

- *Cast-as-intended.* It may be verified that the encryption or other representation of the vote is indeed a representation of the proper vote. Methods to ensure cast-as-intended behavior include Chaum's work [XXX] or the challenge/repsonse method by Neff [YYY].

- *Recorded-as-cast.* It may be verified by the voter that her vote is properly taken into account. For example, voters may be given identifiers, and a public bulletin board may contain a listing of voter identifiers and corresponding encrypted votes. We must ensure that this step does not reveal the plaintext vote to an adversary.

- *Tallied-as-recorded.* It may be verified by anyone that the final election result is correct given the individual votes. For example, a list of encrypted votes would be homomorphically added, and then the election officials would provide

a zero-knowledge proof that the election result corresponds to a decryption of the tallied votes without revealing the decryption key.

1.4 Conclusion

In this chapter, we presented an overview of voting systems: their history, implementations, and downfalls. We discussed the various notions of privacy and verifiability and introduced the concept of an end-to-end verifiable voting system. In the remainder of this thesis, we survey the relevant technologies used in end-to-end verifiable voting systems in Chapter 2, introduce the split-value system and its implementation in Chapter 3, present performance optimizations to the system in Chapter 4, discuss opportunities for future work in Chapter 5, and conclude in Chapter 6.

Chapter 2

Electronic Voting Systems and Technologies

Chapter 3

End-to-End Verifiable Voting using Split-Value Representations

In this section, we present the split-value system enabling end-to-end verifiable voting as described by Rabin and Rivest [CITE] as well as an overview of the implementation as given in [CITE MARCO'S THESIS]. We will focus on the main ideas and components of the design and implementation of the system, and leave the explanation of the details to the respective papers [X, Y].

3.1 The Split-Value Voting System Design

3.1.1 Definitions

Given an election, choose the *representation modulo* M so that any voter's selection, including possible write-in candidates, may be represented by an integer modulo M.

Definition 3.1.1. The *split-value representation* of x modulo M is the pair $\mathrm{SV}(x) = (u, v)$ such that $x = u + v \bmod M$.

Note that for a given x, knowing the value of one of u or v, but not both, reveals no information about x. This key property allows us to construct zero-knowledge proofs of equality for two values using their split-value representations, without revealing the actual values themselves.

Definition 3.1.2. A *commitment* to a value x with randomness r is the value $c = \textsc{Com}(x, r)$ such that c may be "opened" to reveal x and r. We desire that the Com function is both hiding and binding. That is, given $c = \textsc{Com}(x, r)$, no information is gained about x, and it is infeasible to produce another (x', r') such that $c = \textsc{Com}(x', r')$.

The HMAC function, with the randomness r as the key, is used as the commitment function in our implementation.

Definition 3.1.3. Given $\text{SV}(x) = (u, v)$ and randomness r and s, the *split-value commitment* of x is $\textsc{ComSV}(x) = (\textsc{Com}(u, r), \textsc{Com}(v, s))$.

Using split-value commitments, we may construct probabilistic proofs that two values are equal without revealing the value itself. Suppose that a prover wishes to prove that $x = x'$. Given their split value commitments

$$\textsc{ComSV}(x) = (\textsc{Com}(u, r), \textsc{Com}(v, s))$$
$$\textsc{ComSV}(x') = (\textsc{Com}(u', r'), \textsc{Com}(v', s'))$$

the prover claims that there exists a value t such that $t = u - u'$ and $t = v' - v$. This can be true if and only if $x = x'$. With probability $1/2$, the examiner asks for the opening of the "left" commitments, in which case the prover gives the values u, r, u', and r'. The verifier checks that the values $\textsc{Com}(u, r)$ and $\textsc{Com}(u', r')$ are correct and that $t = u - u'$. With probability $1/2$, the examiner asks for the "right" commitments, in which case the prover sends the other components of the split-value commitment. Thus, if $x \neq x'$, a dishonest prover can win with probability at most $1/2$.

3.1.2 The Verifiable Mixnet

The main component of the split-value system is a verifiable mixnet composed of a 3×3 grid of mixservers. These servers are responsible for receiving votes, obfuscating and permuting them, and producing an output list that cannot be traced back to the input list. Denote the mixserver in row i and column j as $P_{i,j}$.

When the voter's vote w is cast on her device (e.g. a tablet), it is broken into three components x, y, and z such that $w = x + y + z \bmod M$. The device then creates split-value representations of each component. For each component, the split-value representation and its commitment are sent over a secure channel to the first column mixserver of the row corresponding to the component. For example, x and $\textsc{ComSV}(x)$ are sent to $P_{1,1}$, y and $\textsc{ComSV}(y)$ are sent to $P_{2,1}$, and so on. Note that because each mixserver only receives one out of three components of the vote, no single server may reconstruct any individual vote (and break voter privacy) without leaked information from at least two other mixservers.

Definition 3.1.4. A triplet $S' = (x', y', z')$ is an *obfuscation* of $S = (x, y, z)$ if $x' + y' + z' = x + y + z \bmod M$.

The first column of servers agree upon a triplet of values (p, q, r) such that $p + q + r = 0 \bmod M$ for each vote. The first row then computes $x' = x + p \bmod M$, the second row computes $y' = y + q \bmod M$, and so on. The column of servers then agree upon a random permutation $\pi : \{1, \cdots, n\} \to \{1, \cdots, n\}$. Each server transmits the permuted obfuscated values to the next column in the same row: $P_{1,1}$ transmits $x'_{\pi(1)}, \cdots, x'_{\pi(n)}$ to $P_{1,2}$, $P_{2,1}$ transmits the permuted y' values to $P_{2,2}$, and so on.

The second and third columns of mixservers repeat this process, and the results of the final obfuscation and permutation are posted to the secure bulletin board. Given a security parameter m, this process of obfuscation and permutation across each column of the mixnet is performed $2m$ times. In our current implementation, $2m = 24$.

To verify the election result, we divide the $2m$ repetitions into two groups of m. The first m repetitions are used to show that the mixnet does not alter the votes. To do so, we reveal the permutations used at each column. For each split-value commitment on the input side of the mixnet, we follow the revealed permutations to arrive at its corresponding split-value commitment on the output side and construct a proof of equality as described in the previous section. These permutations and proofs of equality are posted to the secure bulletin board for anyone to examine.

The other m repetitions are used to prove that the election outcome is correct. All of the commitments to the votes are opened, revealing the votes themselves. These votes are posted to the secure bulletin board along with the election outcome for anyone to examine.

3.2 Implementation Details

The original design of the system included a functioning preliminary prototype written using Python 3.4. In this prototype, mixservers were simulated using Python objects. A simulation of a complete election, including vote generation, mixing, proof construction, and verification, was contained within a single Python process. Python was chosen for its ease of programming, debugging, profiling, and installation. Care was taken to ensure that included libraries was kept to a minimum, so that the code was easy to run and results easily replicated.

3.2.1 Server Processes and Communication

Subsequent work [CITE] significantly decoupled the system. The mixservers, secure bulletin board, and vote generation each became its own process. An interface was built on top of Python TCP socket servers to facilitate communication, and Python socket handlers were used to handle incoming requests. All data was serialized with standard JSON libraries. The implementation also introduced the notion of a *controller server*, whose responsibilities include the following.

- Serve as the first point of contact for all other servers. All other servers start knowing the IP address and port of the controller.
- Keep track of and broadcast network information, including all other servers, their addresses, and roles, to the network.
- Assign roles to each other server, and assign their positions in the mixnet.

- Synchronize the various phases of the election simulation, including vote production, mixing, proof construction, tallying, and verification.

Because all communication occurs via standard TCP sockets, the implementation may be used on any group of computers running on a local area network. The implementation has been tested on single-core and multicore machines as well as multiple virtual machines running on a virtualized network.

3.2.2 The Implementation of Randomness

One particular detail that deserves special attention is the implementation of randomness. Randomness is used throughout the system: in constructing split-value representations, in constructing commitments, in choosing which component of a split-value representation to reveal during a verification, and so on. Fast, cryptographically secure randomness is essential to the correct functioning of the system.

The utility function library written for this implementation includes a function that generates pseudorandom bytes. Each server maintains a list of randomness sources. When a randomness source is created, its name is used as the initial seed. When a sequence of pseudorandom bytes is desired, the seed is hashed twice using the SHA-256 hash function. The first hash determines the updated seed, which is stored in the place of the original seed. The second hash is of the original seed with a "tweak" string appended. With a good hash function, these two hashes will not appear to have originated from the same seed.

During the production of the proofs, we require a good source of randomness to choose the m repetitions that will be used to verify the mixnet and the other m repetitions that will be used to verify the election result. There are two options for this.

- An external source of randomness, such as a dice-roll which occurs at a public ceremony after voting has ended. The benefit to this method is that we obtain "true" randomness. However, it is possible that election officials and the public

are uncomfortable with introducing what appears to be games of chance into the election process.

- A Fiat-Shamir style hash of the secure bulletin board. With overwhelming probability, a Fiat-Shamir style hash of the secure bulletin board will provide randomness that is good enough to overcome an adversary. There is the small but nonzero chance that an adversary could manipulate the election result by providing both a faulty proof that supports the manipulated outcome and an alternate version of the secure bulletin board that, when hashed, produces randomness that verifies her faulty proof. To compensate for this, we set the number of repetitions (m) to be higher so that the chance of producing an alternate secure bulletin board that passes all of the verifications decreases exponentially.

In the current implementation, we use the Fiat-Shamir method. However, this choice incurs a penalty in performance, as our setting for m could be reduced dramatically by using an external source of randomness, resulting in fewer iterations through the mixnet and a shorter proof. In the next chapter, we will see that the use of an external source of randomness may provide enough of a performance boost to meet certain goals that election officials desire.

3.3 Conclusion

In this section, we introduced the concepts and design behind the split-value verifiable voting system, including the idea of a split-value commitment. We discussed the ways that votes are obfuscated and permuted through the mixnet. Proofs are constructed to verify both the mixnet and the election outcome without revealing associating voters with their votes. Finally, we examined details of the prototype implementation, including the communication interface between servers and the implementation of randomness used in our simulations.

Chapter 4

Performance Optimization of the Split-Value System