Spring 2013

Electronic voting: Methods and protocols

Christopher Andrew Collord
James Madison University

Electronic Voting:

Methods and Protocols

Christopher A. Collord

A thesis submitted to the Graduate Faculty of

JAMES MADISON UNIVERSITY

In

Partial Fulfillment of the Requirements

for the degree of

Master of Science

InfoSec 2009 Cohort

May 2013

Dedicated to my parents, Ross and Jane,

my wife Krista, and my faithful companions Osa & Chestnut.

Acknowledgements:

I would like to acknowledge Krista Black, who has always encouraged me to get back on my feet when I was swept off them, and my parents who have always been there for me.

I would also like to thank my dog, Osa, for sitting by my side for countless nights and weekends while I worked on this thesis—even though she may never know why!

Finally, I would also like to thank all who have taught me at James Madison University. I believe that the education I have received will serve me well for many years to come.

Contents

List of Tables.

List of Figures.

Abstract.

The act of casting a ballot during an election cycle has been plagued by a number of problems, both intrinsic and extraneous. The old-fashioned paper ballot solves a number of problems, but creates its own. The clear 21st Century solution is the use of an automated electronic system for collection and tallying of votes, but the attitude of the general populace towards these systems has been overwhelmingly negative, supported in some cases by fraud and abuse.

The purpose of this thesis is to do a broad survey of systems available on the market now (both in industry and academia) and then compare and contrast these systems to an "ideal" system, which we attempt to define. To do this we survey academic and commercial literature from many sources and selected the most popular, current, or interesting of the designs—then compare the relative strengths and weaknesses of these designs.

What we discovered is that devices presented by industry are not only closed-box (which makes them inherently untrustworthy), but also largely inept in security and/or redundancy. Conversely, systems presented by academia are relatively strong in security and redundancy, but lack in ease-of-use or miss helpful features found on industry devices.

To combat these perceived weaknesses, we present a prototype of one system which has not previously been implemented, described in Wang [1]. This system brings together many ideas from academia to solve a significant number of the issues plaguing electronic voting machines. We present this solution in its entirety as open-source software for review by the cryptographic and computer science community. In addition to an electronic voting implementation this solution includes a graphical user interface, a re-encryption mix network, and several decryption methods including threshold decryption. All of these items are described in-depth by this thesis.

However, as we discuss in the conclusion, this solution falls short in some areas as well. We earmark these problem areas for future research and discuss alternate paths forward.

Chapter 1 Introduction

1.1 Foreword

The last hundred years have seen a dramatic change in lifestyle for the average American. Innovations have caused major changes in the way we live, work, and play. Computers and the growth of the Internet have solved problems in ways that even the most forward-thinking minds of the past century could never have envisioned.

Yet, with all these changes, Americans in the 21st century vote for public officials in much the same way their as ancestors did hundreds of years ago. Many systems have been proposed and implemented in an attempt to bring voting into the current century, but quite often these designs suffer from systemic problems--others simply disappear or never see implementation.

In some cases the reason for this is cost, but for many the real answer lies much deeper than the systems themselves. In the United States, voting suffers from a number of very real problems, and at the head of these issues is a matter of trust. Concerns over electoral fraud have become increasingly visible, fueled by the use of "black box" vote counting technologies and media reports of lost or misinterpreted votes—often with implications of fraud. In an electronic system, a single programming error can result in the loss, modification, or addition of thousands or millions of votes. This is a fact not lost on those who wish to maliciously tamper with election results. As Ronnie Dugger famously postulated in 1989, "The next President of the United States may not be chosen by the voters of the United States. Instead, he or she may be the choice of whomever controls or manipulates the computer systems that tally the votes."

What is needed, then, is a vote tallying system that gives voters a high degree of confidence that their vote has been counted as intended, while at the same time providing voters with an insurmountable degree of privacy. The purpose of this thesis is to examine existing designs and leverage their strengths to produce an electronic voting (e-voting) system for use by American voters. A number of very specific requirements must be met for this system, and we will examine these in great depth. Requirements, both technical and physical, are then detailed. A sample implementation of this system is provided. Finally, the sample system is analyzed and conclusions are drawn from the results.

1.2 Definitions of Terms

In a voting system, there exist a number of well-defined roles and objects which must interact. These elements serve as the building blocks for our e-voting solution, so we will briefly define them here.

1.2.1 Voting: Roles and Objects

The following terms and definitions are be used throughout this thesis.

- **Voter:** An individual who has a legal right to vote.
- **Registration Authority:** A voter must be officially registered in order to vote. The registration authority:
    - Determines that a voter is eligible for the right to vote.
    - Ensures that a voter only votes once in a given election.
    - Provides an electronic key or physical ID, if needed.
- **Ballot:** A generic term for the way in which a vote is officially submitted. A ballot contains a vote, and thus we use the two terms interchangeably. When a voter wishes to cast their vote, they do so with a ballot. Whether the ballot is electronic or paper-based is inconsequential.
- **Candidate:** A person who seeks election to, and has a legal eligibility for, a political office. A candidate is elected if he/she is selected on the greatest number of ballots.
- **Polling Location:** A locale where a voter can go to cast their ballot. Each voter is assigned to a specific polling place, based on their physical location. Currently, voters may not cast their ballot from any polling place except the one they are assigned to.
- **Tallying Authority:** Responsible for collecting ballots and tallying election results.

1.2.2 The Phases of Voting

Voting consists of several stages. The order of these processes must be preserved to ensure a valid outcome of the election.

- **Registration:** Voters submit a request to the Registration Authority to vote in upcoming elections.

- **Authorization:** The Registration Authority determines that a voter is eligible for the right to vote. On the day before the election is to be held, a list of eligible voters is released to each polling station.
- **Authentication:** The voter arrives at their polling location on the day of the election. Registration authorities verify that each voter present is on the list of eligible voters, and has not already voted. If this is the case, the user is given a single ballot with which to cast their vote.
- **Voting:** The voter casts his or her vote on the provided ballot.
- **Tallying:** The tallying authority counts the ballots received, and announces the results of the election.

Figure 1.1 depicts the necessary order of the process.

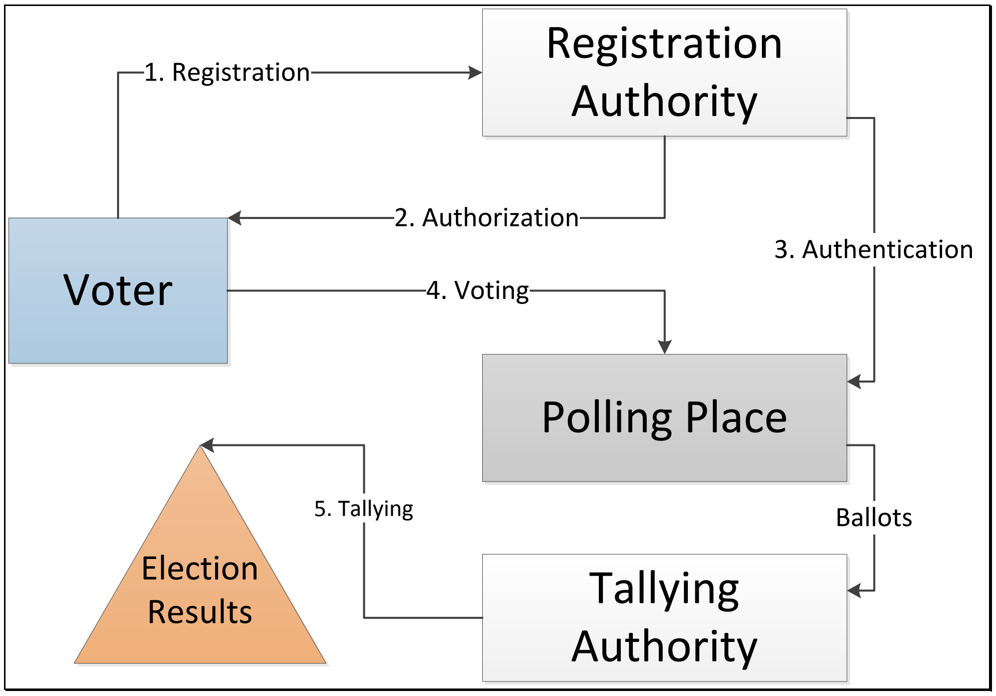


Figure 1.1: Voting Process Flow

1.2.3 Electronic Voting

The term "electronic voting", or e-voting, is used to describe a fully-electronic means of capturing and counting ballots for an election. In its most basic form, electronic voting consists primarily of two devices:

- **Terminal:** A standalone device capable of securely receiving, recording, and transmitting a vote. One of these devices must be present in each voting booth.
- **Supervisor:** One or more centralized servers capable of receiving and storing votes from all terminals. These servers are only accessible by trusted individuals who are responsible for each polling location.

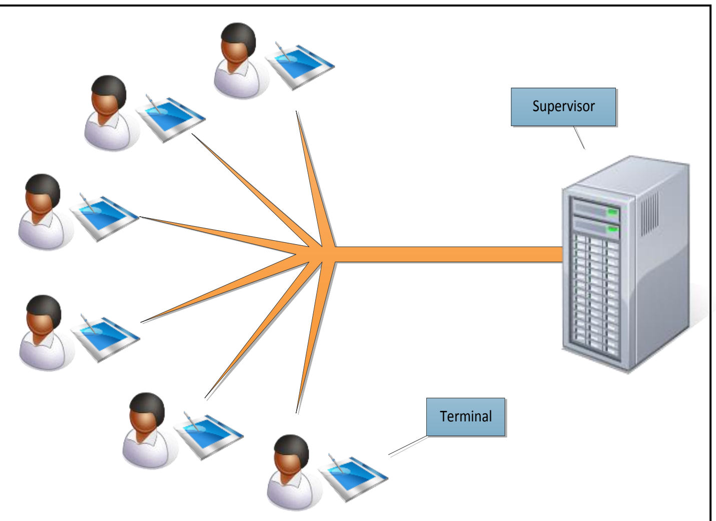


Figure 1.2: Basic e-Voting System

Figure 1.2 shows a basic e-voting scheme. The lines drawn between the Terminals and the Supervisors represent a **network** over which data is transmitted. This network must remain isolated to prevent possible malicious tampering or compromise of privacy.

The e-voting scheme shown in Figure 1.2 suffers from a number of problems, but before we discuss these we must first lay out some terms which are used to define the features of an ideal e-voting scheme and identify possible problems.

1.2.3.1 E-Voting Terms

Electronic voting technology is not meant to completely replace the current election system—only to make the gathering and tallying of votes faster, more reliable, and more secure. Concepts which are unique to electronic voting include:

- **Receipt:** A receipt is tangible evidence that a vote was cast. It is kept by the voter to link them to a ballot, and later used for tracking of their vote.

- **Zero-Knowledge Proofs:** A receipt-less alternative to tracking votes. Zero-knowledge proofs provide a way for a voter to ensure their vote was counted using something that the voter *knows*, such as a word, phrase, or number. We do not be use zero-knowledge proofs in this prototype.

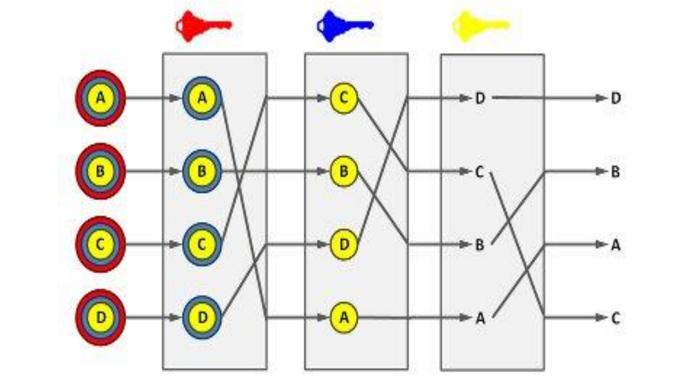


Figure 1.3 A Decryption Mix Network

- **Vote Verifiability:** A voter can track their vote and ensure it was counted as intended, using their receipt.

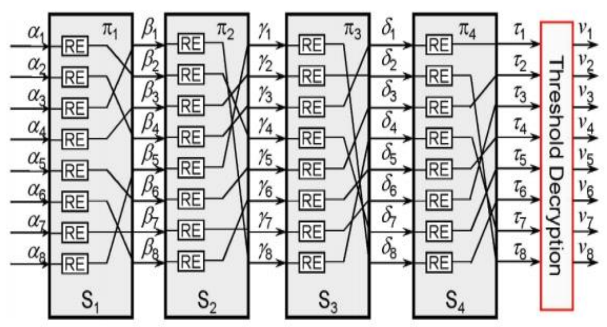


Figure 1.4: A Re-Encryption Mix Network

- **Non-Coercibility:** In spite of the voter being given a receipt (or using a zero-knowledge proof) with which they can track their vote, the voter must not be able to use this evidence to prove to a third party which candidate they selected. If a voter were able to prove how they had voted, a malicious candidate or third party might be able to threaten or otherwise coerce a voter into selecting a particular candidate.

- **Vote Privacy:** A voter's selection must be impossible to determine with more than 50% accuracy, including by election officials. Although the concept is not new, electronic voting presents some unique problems due to vote verifiability. These problems must be solved to ensure voter privacy.

  - **Mix Network:** A "mix network" or "digital mix network" as proposed by David Chaum is a network of independent proxy machines chained together in a specific order, each of which has a unique public/private key pair. When a vote is cast, the vote is encrypted using the public keys of each member of the mix network in reverse order by the terminal, then transferred to the first proxy in the mix network. When a vote is received by a decryption mix, the mix uses its private key to "peel off" a layer of encryption. When the final mix removes the final layer of encryption, the vote arrives unencrypted at the supervisor. This concept is illustrated in Figure 1.3. Decryption mix networks provide security as well as anonymity, because an attacker must know the private keys for each encrypted layer. However, decryption mix networks must maintain a strict order and replacement of a single mix requires that the mix be outfitted with the correct private key.

Wang [1] proposes the use of a mix net, but instead recommends the use of a *re-encryption* mix net. A re-encryption mix network (as shown in Figure 1.4) uses a similar topology and shuffling operation as a decryption mix network, but re-encrypts the message rather than decrypting it. The mixing operations for both types of mix networks can be done in several ways, but Wang [1] suggests a random shuffling permutation be assigned to each mix in the network. So long as this permutation remains a secret, the mix can be considered secure. In this way if a mix is believed to be cheating, it can be forced to reveal half of its input/output permutations to detect cheating with a 50% probability.

Re-encryption mix networks provide anonymity and flexibility—the origin of the data being transmitted is obscured and the replacement of a single mix or an entire network can be done instantaneously. However, secrecy of data being transmitted through a re-encryption mix network is not guaranteed by the mix itself—it must be protected in some other way.

In this paper we suggest an easier and more secure method for mixing inputs to outputs: each mix produces a double-random mix of inputs to outputs for each set of data. It can then be forced to disclose its entire input to output mapping on a particular set of data without exposing any information which could be later used by an attacker. This allows for a 100% probability of detecting a cheating mix. We discuss this implementation in-depth in Chapter 4.

1.2.3.2 E-Voting Challenges

Electronic voting can be used to solve old problems, but it introduces new problems as well. Some of these problems are:

- **Cheating Terminal:** a cheating Terminal is a machine which dishonestly records a voter's candidate choice. For example, if a voter selects "Bob", and the machine displays "Bob", but enters a vote for "Alice", then the machine has cheated.
- **Cheating Supervisor:** Similar to a cheating terminal, a Supervisor could be programmed to skew vote counts towards one candidate over the other. For example, if a supervisor receives 5 votes for "Bob", but counts 1 of these votes as a vote for "Alice", then the Supervisor has cheated.
- **Cheating Proxy:** In the mix network implementation discussed above, a proxy could be programmed to change its output. For example, if an encrypted vote for "Bob" was received on the input for the proxy, but a re-encrypted vote for "Alice" was produced as output, then the proxy has cheated.
- **Vote Buying:** It is conceivable that a candidate or interested party might bribe or threaten a voter into voting for a particular candidate. Therefore we must make certain that the voter cannot prove whom they have voted for—even with a receipt.
- **Vote Modification:** Even if we have an honest Terminal and Supervisor, it must be considered that a malicious insider could tamper with the network or install an inline device to monitor and/or alter votes. A system to detect tampering must be in place.
- **Over-Voting:** An undesirable condition where a voter chooses more candidates than the position allows for. In the case of too many selections for a position, the ballot is discarded.
- **Under-Voting**: A condition where a voter chooses too few candidates for a position. Although under-voting is not desirable, it is allowable.

1.2.3.3 Electronic Voting: Improved

Returning to the simple e-voting scheme shown in Figure 1.2, there are several obvious problems. There is no mechanism to detect a cheating terminal or supervisor. A voter's privacy could be compromised through the timing of their vote, and there is no system to detect outside tampering.

Several modifications are made to enhance the security of the system. A secondary supervisor is added for failover and error-checking, and a mix network is added to the network between the terminals and the supervisors. The mix networks usefulness is considerable; it detects dishonest terminals while preserving anonymity of the user by protecting against timing attacks.

These modifications are shown in Figure 1.5. Although the mix network is shown as a single entity, it is worth noting that its functionality can be spread out across multiple physical locations.

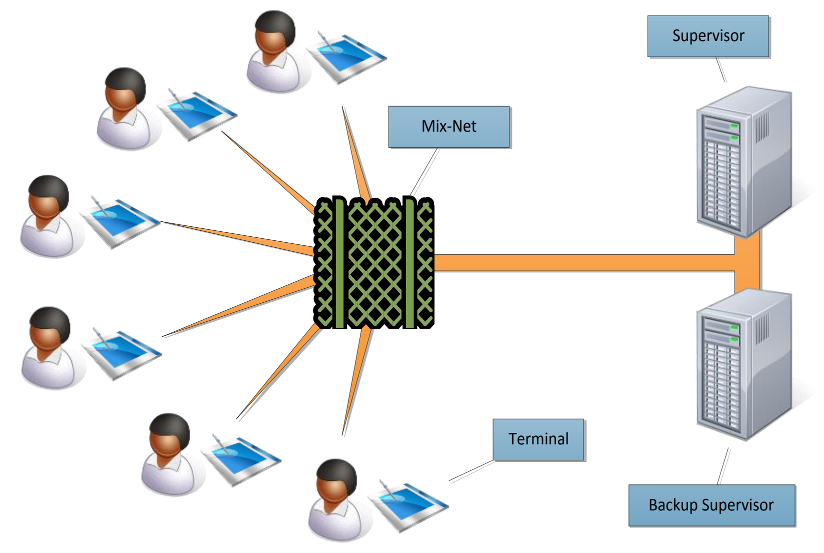


Figure 1.3 e-Voting System Modified with a Mix Network

The individual proxies themselves are also auditable, and a full discussion of how this is done can be found in Chapter 4. During the audit, a proxy in the mix network can be forced to reveal its mixing operations for an individual round of pseudo-votes, which can then be verified on the supervisor. If a vote arrives incorrectly on the supervisor the mix will have been determined to be cheating with a 100% probability.

1.3 Electronic Vote Tallying – Past and Present

Before diving into our proposed system we first look at the two most prominent systems currently in use. The most notable types of electronic ballot devices are:

- Paper-based system with electronic/optical recognition tallying, sometimes called a "document ballot voting system".

- Direct Recording Electronic Voting systems, or DREs.

Each of these systems has a distinct set of strengths and weaknesses.

1.3.1 Document Ballot Voting System

The document ballot voting system is a modification to paper-based voting system which allows for votes to be tallied more efficiently, while removing the human element from the count. The basic system has been in existence for nearly 50 years through many different implementations and often involves an optical or mechanical device to determine how a punch card is marked.

The system first saw use in the Georgia counties of Fulton and De Kalb, during the September 1964 primary elections. The vote was held using punch cards that were electronically tallied. That year in November, the same system was used to tally the Presidential election in Lane County, Oregon and San Joaquin and Monterey Counties, in California (Saltman 1975). As of 2006, about 40 percent of votes were tallied using optical scan systems ("Direct Recording Electronic Voting Machines", 2006).

1.3.1.1 Strengths

Electronically tallied paper systems are very popular to this day. Voting equipment is cheap and standardized, and most counties have owned their equipment outright for decades. If one large county purchases equipment, smaller surrounding counties can make use of this equipment for their ballots. Most importantly, if a system is suspected of cheating or providing incorrect output, a recount can be done manually—paper provides a tangible audit trail which most Americans feel comfortable relying upon.

Paper systems are also incredibly simple. They require only a card and a device for making a mark, hole, or dimple. They are not susceptible to loss of power or other environmental conditions. Cards are easily numbered to prevent lost votes, and cards are nearly impossible to modify once marked—especially on a large scale.

1.3.1.2 Weaknesses

Paper-based voting has drawbacks which make it far from ideal. The Florida "chads" incident of 2000 is a clear demonstration of the failures of paper-based systems. Florida's voting system, which consisted of manual punch cards read by automated vote-counting machines, discounted nearly 2 million American votes in the 2000 Presidential Election. Punch cards were found to have "hanging", "pregnant", or "dimpled" chads—in many cases disqualifying the voter as having "over-voted" (multiple candidates for a single position) or being left blank. Multiple manual recounts were ordered, and the final results determined the 43rd President of the United States to be George W. Bush by 537 votes (Harris 2010). As a result of the subsequent uproar, the Help America Vote Act (HAVA) of 2002 was passed, with over $3 billion dedicated to developing and implementing a modern voting solution.

In addition to these pitfalls, paper cards cannot be traced back to their owners, which is a negative aspect for voters who wish to check that their vote was received as intended. Floridians who participated in the 2000 election have no way of knowing whether their vote counted. Also, the electronic tallying machines used to count paper votes are susceptible to the same forms of tampering as their full-electric counterparts, discussed later. In fact, these devices may be less likely to be caught cheating except in the most egregious ways. Small-scale modification could easily go unnoticed unless a manual recount was ordered, because there is no system to double-check a large number of ballots.

1.3.2 Direct Recording Electronic Voting Systems

Direct Recording Electronic Voting systems, or DRE's, are a paperless alternative to the document ballot voting system. A DRE is an electronic machine which uses a screen to interact with voters and to register votes, which it then commits directly to memory. As such, they make fast and reliable vote tallying possible. Many currently implemented DRE's are simply a digitized version of the traditional paper ballot-based voting system (Wang, 2011). An additional benefit of DRE systems is that they comply with the Federal requirement imposed by the Help Americans Vote Act (HAVA) of 2002 for assisting disabled voters ("Direct Recording Electronic Voting Machines", 2006). DRE's are the most modern voting systems available at this time, yet their lack of standardization and testing makes their future uncertain.

1.3.2.1 Strengths

DRE's have a number of advantages which can be expected from any electronic voting system. They allow for nearly instantaneous tabulation of votes. They are intuitive to use and make allowances for voters with disabilities. The ordering of candidates' names can be randomized to prevent perceived benefits (for example, some studies suggest a candidate whose name appears first in the list may be more likely to receive votes). DRE's also completely prevent the occurrence of conditions such as over-voting, a case in which a voter selects two or more candidates for a single position—and whose vote is thus discarded.

Because of these advantages, DRE's have achieved a fairly large market share. Nearly 25% of voters recorded their vote using a DRE of one type or another in 2006 [2]. This number is likely to increase as more voters become accustomed to the technology, and the price for these devices comes down.

1.3.2.2 Weaknesses

Perhaps the greatest fault of DRE's is that they rely heavily upon the principle of security through obscurity. DRE's are produced by a variety of manufacturers and have equally varied interfaces and voting procedures, and most have not been thoroughly tested by third parties. There is no guarantee that these devices are secure or that their logic is sound, aside from the guarantee given by the device manufacturer. Additionally, these devices are often expensive and failure-prone [2]. DRE's are also impacted by power-related issues such as blackouts, surges, and solar flares—issues which do not affect paper ballots.

An even greater obstacle with regard to widespread deployment of these devices is the lack of tangible evidence in case of malfunction and/or tampering. With a vote that exists only electronically, it is theoretically possible for the vote to be modified at any of several stages during the voting process. Because the voter receives no evidence of how their vote was actually entered into the tally (other than any evidence produced by the same—possibly tampered with—machine), it is impossible to determine that their vote was cast as it was intended. Several modifications have been proposed to rectify this condition, yet none have been fully implemented or accepted.

During the 2004 US Presidential elections in Ohio, concerns about DRE's became increasingly visible. A direct recording electronic machine error gave incumbent George W. Bush 3,893 extra votes when only 638 voters in total had cast their votes in the precinct (McCarthy 2004). The system manufacturer cited an unspecified software error, yet concerns about the companies' political affiliations made many voters uneasy. For many voters, this signaled a need to return to paper-based ballot systems.

1.4 Literature Review

Ideas were borrowed from the following sources to create a trustworthy and robust voting system.

[3]

Chaum details a method of distributing voting receipts. In this scenario a voter electronically selects a candidate, after which the voting machine produces a two-part printout. The printout allows the voter to visually verify their intended candidate. The method uses hundreds of squares consisting of four dots (two black, two transparent), which, when combined with a similar yet different paper, visually spell out the candidate the voter has selected. The voter then chooses to keep either the top or the bottom copy and shred the other while at the voting precinct.

The suggestions made in this thesis are both interesting and have a strong cryptographic backing which ties in well with other ideas for electronic voting. This idea essentially makes use of a onetime pad, which is unconditionally secure IF the pad is destroyed on the source machine and never re-used. Other interesting points made are for the possibility of voting outside ones' precinct.

[4]

A non-technical paper which has significant and interesting statistics, including a case study relating to voter turnout in the US. It effectively illustrates the point that making voting *easier* increases voter turnout in all cases. Because electronic voting has great potential to make voting easier, we believe this paper can provide a strong argument for moving to electronic voting.

[5]

This paper introduces the concept of "Bingo Voting". The basic idea is this: for each voter, each candidate is given a random number by a "*trusted* random number generator" (the authors suggest a bingo-style machine, but note that a less random generator could be used if its algorithm was not known). This results in a large number of "dummy" numbers.

When a voter enters the booth, an unused random number from the *trusted* pool is assigned to each candidate. The voter selects a candidate from an electronic screen, along with a corresponding trusted random number. The machine then generates a fresh set of random, garbage numbers. The chosen candidate retains his unique trusted number, while the other candidates are given garbage numbers. The voter is then tasked to verify that the trusted number next to their selected candidate matches with the trusted number provided by the trusted random number generator.

The authors of this paper are credible and their solution has a number of advantages, however the disadvantages outweigh them. Complexity is the majority problem, with voters asked to verify (possibly large) numbers for their candidate.

The paper does make an interesting suggestion with regards to vote tallying in the form of *receipt hashing*. The authors note it could be applied to many other forms of voting as well. The general idea is to provide a hash with the first receipt, then base the hash of the second receipt on the first, the third on the second, etc. In this way the hashes are recoverable by starting from the last receipt and decrypting back until the hash for the first receipt is verified. Missing vote cards would be immediately noticed. A downside to this method is that a clear timeline could be established—for example we would know exactly which vote card belonged to the last voter, resulting in a clear violation of privacy. If votes were submitted with a random delay, however, the idea could prove very useful.

[6]

Chaum, et al. suggest another method for providing a "receipt" to voters whose votes are electronically tallied. The idea is not to provide a completely new system, but rather a small modification to existing systems which allows voters to track their vote and ensure that it gets counted. The basic change is to add a unique serial # ID to each ballot, half of which is torn off and retained by the voter. The receipt allows for verification without providing proof of which way a voter cast their vote.

Although this system does not do away with the traditional paper ballot/electronic scan system, it does provide an additional level of verification, and it does so at a low additional cost, and very low complexity level.

[7]

Fan et al. provide a broad overview of existing voting technologies, including desirable and undesirable traits, strengths and weaknesses, and possible modifications. The authors introduce a new system which they call Lei-Fan. The system is receipt-free—the author believes receipts result in coercibility—and instead makes use of "zero-knowledge proofs".

These zero-knowledge proofs require two additional concepts: dual randomization (the user selects a string and combines it with a randomly generated string) and multiple-receipts (wherein after casting their vote, the voter can receive both their receipt and the receipts of any other voter at the polling place—thus it is impossible to prove which receipt belongs to which voter).

Although the concepts are interesting, they are excessively complex. The implementation suggested requires heavy computation, and the idea of multiple receipts is not as user-friendly as some solutions by Rivest, Chaum, etc.

[1]

Wang, et al. provide an overview of the problems with electronic voting, and cover Rivest's proposed mix network solution. They suggest using a semantically secure modification of the ElGamal encryption scheme.

Possible problems with the model are noted at the end of the paper, and I believe that several of these could be improved if combined with a model like the one proposed by [3]. Receipts are necessary but asking a voter to verify that one number is identical to another is much less efficient than asking them to verify text.

[8]

Secondary source. Wolfinger has quite a few articles relating to American voting policies and this paper makes a few excellent points regarding voter turnout. This paper suggests a new method for calculating voter turnout, as well as providing several statistics about who votes and why.

[9]

Early NIST paper discussing the use of electronics for vote tabulation. Outdated, but very in-depth and contains good information about problems and solutions that have been found with systems. Also gives historical information.

[10]

Broad overview of electronic voting machines circa 2010. Mentions the "chads" incident and what has been done to improve the systems since then. Very few technical details but provides a good overview. Mentions several startup companies and the difficulties they have faced, with money being a primary issue.

Chapter 2 Functional & Security Requirements

In this chapter we describe an optimal electronic voting machine. Requirements for an electronic voting machine can be broken into two categories. The first category is *functional* requirements. These requirements detail exactly what a device must do in order to serve its purpose as an electronic vote recording machine. We then look at the *security* requirements for such a machine. These requirements detail what a voting machine must do to insure that data is not added, altered, or lost at any point during the voting or tabulation process and to ensure that security functionality is transparent to the casual voter.

2.1 Functional Requirements

An ideal electronic voting machine meets the following functional requirements. In later sections we deal with the areas in which some of these requirements conflict.

[1] **Tally Correctness:** The vote quantity and candidate selection input into the device is *exactly* the same as the output.

[2] **Convenience:** Voters should not be constrained to a specific precinct, but allowed to vote in whatever precinct is most accessible to them. Note that local laws may require a voter to only vote in their specific precinct, so this is not an automatic disqualifier for an e-voting machine.

[3] **Simplicity:** System must be simple enough that any user can use it. The primary functional goal of an electronic voting machine is to make voting easy for the full spectrum of citizens. Special care must be taken that the interface is intuitive, clean, and easily understandable.

[4] **Vote Verifiability:** The system must produce tangible evidence of a vote. The system produces or relies upon a *tangible piece of evidence* which could later be used by the voter to verify that their vote was submitted and counted correctly. If a machine was suspected of cheating, this receipt may be used by authorities to discover and track modified votes.

[5] **Ballot Secrecy:** A voter must *not* be able to prove to a third party which candidate he or she selected. This is done to prevent vote selling or coercion by any outside party.

[6] **Anonymity of Vote:** Votes cast at a precinct must be indistinguishable from one another—it must not be possible to determine which terminal a vote was cast on, or at what time.

[7] **High Availability:** System must have 100% availability during voting hours. No down time is acceptable. System availability is a critical point of attack which must be defended against. Power outages, broken cords, malfunctioning terminals and the like can create a denial-of-service condition that must be avoided at all costs. Power problems may be mitigated by installing battery backups which can power systems through black- and brown-outs, and must at least have the capacity to power systems long enough for data to be written from memory to some kind of permanent storage, such as a hard disk.

[8] **High Capacity:** Every machine should have sufficient memory to store every vote cast on it. Terminals must store each vote, encrypted, locally, as well as transmit the vote through a mix network to a centralized server. At the end of an election period, a count can be performed on the terminals, and compared to the total number of votes collected by the central server. A difference in the number of votes may indicate a corrupt mix network or a loss during transmission. For this reason each terminal must contain enough memory to store all votes cast on it in a single day. Machines that lack the storage capacity for all the votes they receive have been a consistent problem in prior US elections.

[9] **Candidate Randomization:** Candidates are presented in a random order to prevent perceived advantages. Electronic voting machines have the potential to create unfair advantages for some candidates. One study [11] found that the way information is displayed on voting machines (including layouts, colors, symbols, and the order in which candidates are listed) influences a voters' decision unfairly. Theoretically, it is be possible for a candidate to gain votes by subconsciously tricking voters, using tactics similar to those used in product advertisements. For this reason it is best to provide voters with a randomized list of candidate's names which voters use to make their selection. After a candidate is chosen, the machine provides the selected candidate's name again to allow the voter to verify that the selection was made correctly. The voter then approves or denies the candidate choice. Randomization prevents advantages--perceived or real--gained by a candidate whose name is placed first on the ballot.

[10] **Compliance with Americans with Disabilities Act:** Capacity for audio/braille instructions to accommodate blind, color blind, and illiterate voters. Alternative inputs to facilitate motor skill-impaired voters. Some citizens may be visually or otherwise handicapped, and it is of utmost importance that these needs are accommodated.

2.2 Security Requirements

Vote verifiability and ballot secrecy are difficult problems that are addressed by security requirements. They are conflicting requirements—the need for secrecy makes verifiability a difficult problem. Verifiability and secrecy are also, arguably, the most important requirements for a successful e-voting system. To provide complete functionality, the system must be designed in such a way that a voter is able to verify their vote was entered into the system without being modified or lost—yet at the same time it must be impossible for them to prove to a third party which candidate they selected. This prevents the possibility of vote-selling or coercion, while at the same time providing voters with peace of mind that the system is functioning properly and their vote was counted.

An electronic voting machine must adhere to the following security requirements.

[11] **Unique-Appearing Ballots:** All ballots should be unique from one another in their encrypted form while stored on the Supervisors, indistinguishable until decrypted for the final tally. For example, if a vote for "Bob" is always encrypted as "AAAAAAA", it is possible for an election official to determine how an individual voted simply by viewing votes stored on the Supervisor.

[12] **Full Encryption of Ballot:** The voter's session must be encrypted at the start of the session, at rest in memory or on hard disk, and while being transmitted to the central server. We must assume there is a possibility of an attacker gaining remote access to a voting machine. Hypothetically, an attacker could modify the vote cast, or implement a routine that modifies all votes from a terminal. By enforcing encryption at all times, there is a high probability that any modification to the session will result in a garbage vote, which will be noticed with a high probability by monitoring the equipment. The session must be encrypted in memory during its creation, and the vote must be encrypted while at rest on the terminal—whether in memory or on a permanent disk. It must also be encrypted

during transportation to the central logging server through the mix net. If, at any time, the unencrypted vote is visible, an attacker has an opportunity to strip the voter of their privacy or to modify the vote.

[13] **Semantically Secure Encryption:** Elections are highly visible and require top-tier encryption. The incentive for an attacker is great, so the system must use the strongest encryption available: a semantically secure encryption algorithm. "Semantically secure" means that a computationally bounded attacker (e.g. one who does not have access to unlimited computing resources) cannot determine which of two encrypted messages corresponds to a known plaintext message with greater than 50% probability. However, few semantically secure encryption algorithms exist. The Goldwasser-Micali [12] and Paillier [13] cryptosystems are considered semantically secure, and padding can be used to make other encryption algorithms such as RSA semantically secure. Wang [1] suggests a simple modification to the ElGamal cipher which makes it semantically secure. The semantically secure algorithm chosen for our implementation is the modified ElGamal cipher, and we discuss in Chapter 4.

[14] **Detection of Cheating Machines:** If a system is cheating, there must be a probability that it will be caught. These probabilities will add up over successive votes, thus increasing the likelihood that a machine is caught. It may be very difficult to detect a machine that cheats only once—however, if multiple machines are cheating, or a machine cheats multiple times, its likelihood of being caught must be significantly greater.

[15] **Secure Decryption of Votes:** No single user should be capable of decrypting all votes. Rather, several individuals must collaborate to decrypt votes. The possibility of a malicious insider must be assumed. If a single individual (e.g. the manager of a precinct) were able to decrypt all votes cast within a precinct, that individual would have the opportunity to strip individuals of their privacy, or to report vote tallies incorrectly. For this reason it is important that decryption of the ballot requires multiple people. One such way to do this is using threshold decryption as described in [1]. Using Shamir Secret Sharing, a private key can be split into several pieces. The ballot is then decrypted in stages, without ever fully reconstructing the full private key. A full description of threshold decryption using Shamir secret sharing is included in Chapter 4.

[16] **Open Source:** Ballot software source code should be available to the public, free of charge. Although this seems like an odd requirement, it serves several important purposes. Making the program open source allows it to be presented to the general public several months prior to the election. Each political party should obtain an expert to view the code and ensure that it does not treat one party more favorably than another. Any member of the general public who has the knowledge and interest may also view the source code, increasing the general public's trust of the system. Once the code has been approved, an SHA1 and MD5 hash of a known-good machine should be taken, and all machines must present an identical hash of their operating and voting systems prior to being reported as fit for use.

[17] **Voter Privacy:** A voter's candidate choice should be protected from all individuals, even those involved in the voting process. Ballots must not be able to be linked to an individual, nor should an individual be able to be linked to a specific ballot.

Chapter 3 Modern Implementations

In this chapter we look at modern implementations of e-voting solutions. Each system in this section has been chosen because of its popularity or unique features. These systems are then analyzed to determine whether they do or do not meet the criteria laid out in Chapter 2. The results are then compiled into a spreadsheet showing success (device meets criteria) or failure (device does not meet criteria).

3.1 Election Systems & Software: iVotronic®

Election Systems & Software (ES&S) is an e-voting system designer based in Omaha, Nebraska. ES&S has been in business for approximately 35 years, and recently bought their largest competitor, AccuVote (a division of Diebold). Although they manufacture approximately 10 different types of machines for vote tabulation, their core focus is on paper ballot generation and counting. The iVotronic® is their only all-electronic vote recording system, and it is available both with and without a paper audit trail.

The iVotronic® has an intuitive interface--the company's website allows you to interact with a simulated machine, which is impressively simple. The machine allows multiple languages and has audio assist and braille navigation for Americans with Disabilities Act compliance. Because the machine is completely self-sufficient, it can be used curbside or placed in a handicap-accessible location. The 15" touch screen in easy to read. Each system has "three independent but redundant" processor/memory paths to prevent vote loss, and each individual ballot is stored fully on the machine in case the election is contested.

Those features aside, the iVotronic® is quite distant from the optimal voting system described above. The system is non-networked—instead, each device serves as both the terminal and supervisor. In the case of a complete machine failure, all votes on the individual machine are lost (unless the precinct is using the version with a paper audit trail). The system is closed-source and makes no mention of external auditing. Voters are given no receipt, and no method for tracking their vote.

A literature search regarding this model turned up even worse results in the real world. States such as Colorado, California, Ohio, and Florida have abandoned the technology due to repeated problems and accuracy concerns [14]. A report by the New Jersey Institute of Technology [15] cites numerous failures in the ES&S iVotronic® in their 14-hour testing period, including 3 printer jams on a system with a paper audit trail—one of which was severe enough to tear the paper.

The system is provided as a black box, and the internal workings are not public knowledge, so we were forced to place question marks in the chart for certain requirements. This system does not meet the criteria an optimal e-voting solution.

3.2 Hart eSlate

The eSlate by Hart Intercivic is another USA-based e-voting machine manufacturer. Rather than a touch screen, the device makes use of a scroll wheel and selection button—the voter scrolls through the available options, then selects a box by clicking the selection button when their candidate is highlighted. Although the interactive display on their website felt more cumbersome than the touch screens that other manufacturers offer, they rightly point out that the optical selection wheel and button are extremely reliable and much more precise than a touch screen.

The eSlate gets a number of other things right, too. When a vote is cast, it passes down three independent channels to three separate, external storage devices. Although the operating system of the device is not specifically named, the manufacturer touts it as a "real time operating system" that is "… used in critical medical applications, precision laboratory measurement devices …"

The eSlate also employs physical *and* electronic intrusion detection controls, something not seen on other devices. It uses a proprietary database system which is protected by encrypted password, and core software on the device is never changed—only the election data can be changed. The devices are solid state, and not susceptible to magnetic fields or "abusive handling". The device offers full support for disabled users. Finally, the eSlate has a battery backup which is capable of powering the device for up to 18 hours, or 4 hours longer than an election day.

The manufacturer does not specifically name the cryptographic protocols used, but they state that all data blocks and elements use error-checking techniques for reading and writing of data. Checks are also in place to ensure only authorized systems are on the network, and that all data being communicated "has integrity". Auditing and reconciliation of votes cast is possible too.

The machine falls short in several areas. For one, it does not provide an option for receipts, and it appears that it never will. Further, there is no guarantee of voter privacy, and the device will always be a "closed box"—secure until proven otherwise, for as little as that means. In all, this device is a very good attempt but falls short of meeting our expectations for an optimal e-voting solution.

3.3 Vote Box

Unlike the previous systems, VoteBox is a prototype voting machine and is not currently in production by any business. This makes discussion of its tangible assets impossible; however the prototype does include a UI [16]. In spite of the fact that it is not a production system, VoteBox makes use of some interesting technologies and we feel it is worth discussing. It has received praise in several articles, including [10].

The VoteBox system essentially makes use of a peer-to-peer network over which encrypted votes are broadcast. The network has no external connectivity, and uses two supervisors (one primary, one backup) to record votes. When a voter enters their candidate selection, the VoteBox broadcasts this decision to *all* other machines on the network. After entering their decision, the voter is given a choice to either accept their decision (in which case a separate message is broadcast indicating that the voter has cast their ballot), or to "challenge" the system. In either case, the message is broadcast <u>before</u> the system knows whether the ballot will be cast or challenged. When a system is challenged, the system reveals a key that is used to decrypt the vote and verify its correctness. A vote that is challenged is then lost and must be re-submitted, but the threat of a challenge is used to keep the system honest.

Other interesting technologies include homomorphic tallying (a third party can tally votes without knowledge of the vote plaintexts) and non-interactive zero knowledge proofs (NIZK – a proof that each vote in its encrypted form is a valid vote). VoteBox makes use of a bulletin board for public postings of vote tallies as the votes are cast. The system uses an "additively homomorphic" variant of the ElGamal cipher in which votes are successively chained together.

The creators of VoteBox point out that there is no system to protect voter privacy within the system. If a machine were maliciously tampered with, they point out that it could be configured to record votes as it was broadcasting them, violating voters privacy. Another downside is that the creators have included no verifiability system; instead, they recommend implementing systems similar to PunchScan, Scantegrity, etc.

Although the current implementation of this system does not exactly meet our requirements for an optimal e-voting solution, it is a promising solution that could easily be modified to meet these requirements.

3.4 Prêt à Voter

Prêt à Voter is also a prototype system, currently in its "Version 1" stage. Prêt à Voter reverses the order of events found in our optimal e-voting solution, though at no apparent detriment to the system. Rather than generating a receipt at the end of the voter's session, the receipt is present from the beginning as a physical paper ballot.

Upon entering the polling place and proving eligibility to vote, the voter chooses a paper ballot from a ballot box. The paper is divided vertically into two halves: one half contains the list of candidates, while the other half contains check boxes which correspond with the names of the candidates, as well as a unique code which serves as a key for the specific ballot. A sample ballot is shown in Figure 3.1.

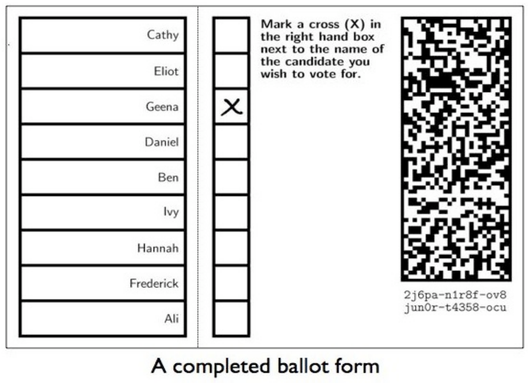


Figure 3.1 Sample Prêt à Voter Ballot

If the voter would like an audit of their ballot, they can simply separate the two halves and enter the right half of the ballot (which contains only check boxes and a scan code) into the machine. The machine prints a receipt showing the candidate names corresponding to the boxes, which can be visually checked against the left half of the ballot. If the two halves match, voters know the machine is being honest. This ballot is now invalid, and the voter is required to start over with a new ballot.

Once the voter has marked the candidates they wish to vote for, the ballot is separated into two halves. The left half, containing a randomized list of candidates, is immediately shredded. The right half of the ballot, containing the marks and security code, is entered into the machine. The machine then prints a receipt.

If the marks on the receipt match the marks the voter has made, the ballot is successful and the voter may retain his receipt as evidence of how his or her vote was cast.

The Prêt à Voter system utilizes a shuffling technology in which vote orders are randomized before being decrypted, ensuring voters' privacy. Candidate lists are randomized to prevent perceived advantages, and also as a primary measure of security. Prêt à Voter also uses the concept of a bulletin board where votes can be posted.

In the original 2005 implementation RSA encryption was used for security, but several successive revampings of the technology have included the use of ElGamal or Paillier encryption for enhanced security, as well as adding a homomorphic quality to the votes. Some implementations have suggested combining the positive aspects of Prêt à Voter with aspects of other voting technologies such as Punchscan and Scantegrity, neither of which are discussed here.

The Prêt à Voter system has few perceived downsides which have not been remedied by current revisions. One major downside to this technology is the lack of handicap accessibility. It is unlikely that the system will incorporate audio and braille assistance for visually handicapped voters. However, it is a very close match to our optimal e-voting solution.

System	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
iVotronic®	✓	✗	✓	✗	✓	✗	✓	✓	✗	✓	?	?	?	?	?	✗	?
VoteBox	✓	?	✓	✓[1]	✓[1]	✓	✓	✓	?	?	✓	?	✓	✓	✓	✓	✓
Prêt à Voter	✓	?	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	?[2]	✓	✓	✓	✓
Hart eSlate	✓	?	✓	✗	✗	?	✓	?	✗	✓	?	✓	?	✓	✓	✗	?

Table 1: Analysis of Several Popular Electronic Voting Devices

1: Depends on verification system used, but suggested systems will pass.
2: Depends on encryption type used; original: no, modified: yes.

3.5 Conclusions

Generally speaking the market for electronic voting machines appears to be very "fly by night". In the past 10 years several companies have appeared, disappeared, been bought out, or been publicly shamed into retirement by software reverse-engineers. Much of this action in the US can be attributed to the Help Americans Vote Act (HAVA), which flooded the market with money to purchase modern voting equipment. Electronics giants such as Sequoia and ES&S got their devices on the market quickly and with little thought in order to soak up the money provided by HAVA. Not surprisingly—and very unfortunately—states have been recalling these devices from use and returning to paper ballots. The phenomenon is not purely American: Nedap technologies (Netherlands) has been hit similarly hard in Europe, with complete abandonment of their technology after a hacker demonstrated a total compromise of their hardware and software.

Although it's difficult to determine precisely whether some devices on the market meet or fail criteria due to their closed-source nature, it's certain that none of them meet our full criteria for an optimal e-voting system. It's not surprising then that the industry-provided solutions are lacking in almost all categories, but what is most unfortunate is that none of the solutions provided by academics have been seriously implemented into a tangible solution. As one commenter noted [10], it is hoped that as the money from HAVA dries up, only the low-dollar academic solutions will remain affordable—they are certainly the best options at this time.

Chapter 4 Prototype Electronic Voting Implementation

We now move forward with a prototype implementation of the e-voting system presented by Wang et al [1]. We begin by discussing each piece of the implementation separately and then combine each piece into a prototype system. Section 4.1 details the e-voting system described by Wang and section 4.2 details improvements and modifications to Wang's e-voting solution as proposed by this paper. In section 4.3, we combine these elements into a workflow which represents the events of an actual election cycle. Finally, we discuss the implementation details realized by this thesis.

The goal of this implementation is to create a software package that can be run on any x86 Debian-based Linux distribution. An operating system is not included as part of this implementation, but development work was done using Ubuntu and Mint systems and the intention is for the code to be portable across many architectures. Some features such as user verification and handicap accessibility are not included in this prototype and are marked for later improvement. This prototype is intended only to show the methods and protocols used in the e-voting system, as well as provide a working transmission and receipt system.

4.1 Building Blocks of Secure E-Voting

In this section we cover the requirements set forth by Wang and his co-authors [1]. We begin by documenting cryptographic requirements in a broad sense, then narrowing the scope to focus on the type of variables needed, the generation of these variables, and how these variables are used throughout the project.

4.1.1 Encryption Requirements

A secure encryption algorithm is perhaps the most fundamentally important building block of a secure e-voting solution. However, we must bear in mind that an incorrectly implemented cryptographic system is worse than no system at all. The irreversible qualities of discrete logarithms in a large prime order subgroup provide the basis for security in all cryptographic operations performed here.

To encrypt the ballot we make use of a semantically secure modification of ElGamal. ElGamal is an asymmetric key encryption algorithm which is not semantically secure in its original form. The basis of ElGamal is a trio of large numbers, p, q, and g. The value p is a large *safe* prime, while $q = (p - 1)/2$, which is also a prime; we know that q is prime because by definition a safe prime is a value of the form $2x+1$ where x is also prime. The value g is an element of the finite field F_p with order q. The value g creates a set $\{g, g^2 \bmod p, \ldots, g^q \bmod p\}$ denoted as <g>. A safe prime is used for the basis of ElGamal because it creates a subgroup of large prime order. Other restrictions are placed on safe primes, but it is not our intent to discuss them here. Generation of a safe prime is left to the OpenSSL libraries.

ElGamal is not semantically secure in its original form because an attacker can determine whether an encrypted plaintext was a quadratic residue or not. In a case where one encrypted value was a quadratic residue and another was not, it would be possible for an attacker to determine which plaintext message correlates to which encrypted message. By restricting all messages to be quadratic residues, this distinction is no longer possible. To make the ElGamal encryption semantically secure, we must ensure that:

- g is an element of order q and g is <u>not</u> a generator of F_p
- The message to be encrypted must be in <g>

These values and their creation are discussed in section 4.1.2.

The message, m, is a value that is a member of <g> representing a candidate's name. At the beginning of an election cycle, each candidate is assigned a message identifier m, which is then distributed to all machines. If the value assigned to a candidate happens not to be in <g>, we can instead encrypt the modular additive inverse of that value—however, as we will discuss in section 4.1.2 this is not an optimal solution and our implementation makes use of a slight modification to this scheme.

When a message m is ready to be transmitted, it is encrypted into a pair of values, c_1 and c_2. We discuss the creation and transmission of these values in 4.1.2. These values are indistinguishable from other votes, yet they allow each voter to verify his or her vote at a later time. These encrypted values are simultaneously transmitted as a ballot, written to an append-only bulletin board, and printed for the user to retain. At the end of the election cycle the Election Authority will verify that the number of votes on the bulletin board corresponds to the number of votes on the Supervisor.

The next phase of voting is to pass the encrypted message (c_1, c_2) to the re-encryption mix network. In this implementation, each mix in the mix network is represented by a section of eight TCP ports, starting with a base port number. For example, the default base port for Mix1 (the first mix in the network) is 31000, and each of its ports are numbered 1 – 8, meaning that the first available port is 31001, followed by 31002, and so on up to 31008. Each mix is then configured with the IP address and a base port where will it forward its re-encrypted payload. Once the mix has received data on all eight of its receiving ports, it shuffles, re-encrypts, and transmits the data to the next mix in the mix network. Implementation details on the listening, shuffling, and retransmitting operations—as well as some security features—are covered thoroughly in this chapter.

During the final phase of transmission, the retransmitted votes are received on the supervisor. Receipt of votes is handled by a module called a "Receiver". Similar to a mix, a Receiver has eight receiving ports calculated from a base port. For the Receiver to work, the final mix in the mix network must be programmed with the base port and host IP of the Receiver. When the Receiver receives data, it simply writes it to disk without performing any calculations on it.

Once the election has concluded, the election officials gather the collected votes from the supervisors and are ready to decrypt and tally the votes. Decryption can be done in one of two ways:

- Standard decryption, in which the private key is provided.
- Threshold decryption, in which the private key is split into pieces and several of these pieces are re-combined into a private key which can then be used for standard decryption.

These decryption methods are detailed in 4.1.3 and 4.1.4 and are provided in our implementation.

4.1.2 Encryption Values and Calculations

The basis for encryption is a trio of large values, p, q, and g, which are generated using the OpenSSL libraries. The safest and fastest way to generate a "safe" prime for p is to provide a TRUE (1) value as the third argument to *BN_generate_prime( )*, indicating that we require a safe prime. We then use p to derive another prime q where $q = (p - 1)/2$.

The final value, g, is found by generating a random value h where $1 < h < (p - 1)$. Another value, e, is set to $(p - 1)/q$, which in our case will always equal two. We then calculate ($g = h^e \bmod p$) as shown in method A.2.1 of [17]. Finally we check that the value g is of order q, and is not a generator of F_p. If the generated value g does not meet these requirements, the process is repeated until a suitable value is found. A sample generation of p, q and g values is found in the Appendix.

Primes & Other Large Values		
Variable	**Use**	**Notes:**
p	Prime	"Safe prime", $p = 2q + 1$
q	Large Prime	$q = (p - 1)/2$
g	Prime	Element of finite field F_p with order q. Not a generator of F_p. $g = h^e \bmod p$

Table 2 Primes and Other Values used for Encryption

Another important value is *r*, a random value generated by the terminal. The *r* variable is used to make each ballot indistinguishable from the next, regardless of content (for example, one vote for "Bob Dole" would be visually indistinguishable from another vote for "Bob Dole" and also indistinguishable from a vote for "Bill Clinton"). The *r* value is generated with the *BN_rand_range( )* function, giving an upper range of (q – 1). The value is then tested to verify that the randomly generated number is not equal to 1.

Random Numbers		
Variable	**Use**	**Notes:**
r	Random	Random value chosen by terminal, $1 < r < q$

Table 3 Other Random Numbers

A public and private key pair, to be used by the election authority for encryption of the ballot is created, also using OpenSSL. *BN_rand_range*( ) is used to generate *s*, while Y is calculated as shown in Table 4 Public and Private Key Pairs.

Authority's Public/Private Key Pair		
Variable	**Use**	**Notes:**
s	Private Key	Random number selected by Election Authority, $1 < s < q$
Y	Public Key	$Y = g^s \bmod p$

Table 4 Public and Private Key Pairs

Finally, the message *m* containing the selected candidate's name is ready to be encrypted and is transferred to the first mix in the network using ElGamal encryption:

$$m_{enc} \rightarrow (c_1 = g^r \bmod p,\ c_2 = Y^r \times m \bmod p)$$

When the message reaches the Supervisor, it can be decrypted using the standard decryption method, where *m*' is the recovered plaintext ballot:

$$\mu_1 = c_1^{\,s} \bmod p$$
$$\mu_2 = \mu_1^{-1} \bmod p$$
$$m' = \mu_2 \times c_2 \bmod p$$

4.1.3 Threshold Decryption

As discussed previously, it is undesirable for a single person to hold the only decryption key. Instead, we use threshold decryption in which a key can be split among *n* people. Our prototype system allows for any value of *n* with *t* shares required for decryption, so long as $n > t$. The variables found in Table 5 Decryption Valuesare used in a threshold scheme.

Decryption Values		
Variable	**Use**	**Notes:**
n		The number of shares created.
t		The minimum number of people needed to reconstruct the secret.
a_i	Random	Random numbers, where $0 \leq a_i \leq q-1$.
s_i	Share	Shares given out. $s_i = f_i\ (1 \leq i \leq n) = s + a_1 i + a_2 i^2 + \ldots. + a_{t-1} i^{t-1} \bmod q$

Table 5 Decryption Values

When a share is calculated, each share is given to Election Authority *i* as $<i, s_i>$. When the time comes to decrypt the encrypted ballots, the steps shown in Figure 4.1 are taken. [18]

- For Election Authority official *j*, calculate modified shadows using LaGrange interpolation such that:

$$\alpha_{\pi\beta(s)} = s_i \cdot \prod_{j=1, j\neq s}^{t} \frac{(0 - i_{(j)})}{(i_{(s)} - i_{(j)})}$$

- Calculate the sum of the modified shadows:

$$s = \sum_{s=1}^{t} \alpha_{\pi\beta(s)}$$

- Calculate $\mu_1 = c_1^{s} \bmod p$
- Calculate $\mu_2 = \mu_1^{-1} \bmod p$
- Calculate $m' = \mu_2 \times c_2 \bmod p$

Figure 4.1: Threshold Decryption

4.1.4 Re-Encryption Mix Network

During transmission, the first mix (*mix1*) in the mix network receives an encrypted message, which we call *a*, where $a = (c_1 = g^r \bmod p,\ c_2 = Y^r \times m \bmod p)$. *Mix1* then performs a transformation on message *a* to create *a'*, where $a' = (c_1 \times g^t \bmod p,\ c_2 \times Y^t \bmod p)$. The value *t* is a new random value generated in the same manner as *r*. This value is generated individually at each of the mixes and the value is never intentionally re-used. The transformed message *a'* is then sent to the next mix in the mix network, where the process begins all over again.

Mix Values		
Variable	**Use**	**Notes:**
t	Random	A value generated randomly for each set of data
a	Data	Data received from previous mix or voting terminal
a'	Transformed Data	A transformation of the data which is then passed to the next mix: $a' = (c_1 \times g^t \bmod p, c_2 \times Y^t \bmod p)$

Table 6 Mix Values

In our prototype, five mixes named {*mix1, mix2, mix3, mix4, mix5*} are used. Interprocess communication is done via TCP ports and mixing is done in a cascade topology with *mix*1 connecting to *mix*2 and so on. Each mix contains eight input/output ports. Complete documentation of the prototype is in Section 4.4 and further documentation is found in the source code.

The variables (g, q, p, Y) are generated by the Prevote module and placed into the *crypto/* folder for the prototype implementation, and the mixes draw from these values when performing calculations. In an actual election cycle, generation of these numbers would be handled by the Election Authority, and these values would be distributed to each machine.

4.2 Modifications and Improvements

In this thesis, we suggest and implement several modifications and improvements to Wang's proposal. In some cases, changes were made to eliminate confusion. In other cases, the change creates a more secure system. In this section, we provide an overview of these changes, with technical details discussed later in this chapter.

4.2.1 Candidate Message ID's

Wang et al. do not specify how or when to create a candidate's ID, which is also known as m. It is conceivable that an ID may be assigned as soon as the candidate announces their intent to run for office, or at any time leading up to the opening of the polls.

As a requirement of the e-voting scheme presented here, a candidate's m must be present in <g>. If it is not, Wang notes that the additive inverse of m must be encrypted instead, as the inverse is guaranteed to be in <g>. Although this solution is cryptographically sound, it can become quite confusing for a poll operator. During implementation, we found that the possible inversion of m created a problem when decrypting the vote.

If an election official is expecting to see a vote for candidate "456" but instead receives a vote for the ID "-456", it may create confusion. This is worsened by the fact that the OpenSSL libraries store their extremely large values as hexadecimal representations where inverses may appear completely different from the original value. This is clearly an undesirable situation.

For this reason, we chose to generate the candidate ID's alongside the creation of the other cryptographic values as the first step in the voting process. An ID is generated for a candidate, and if this ID is not in <g>, we generate another value until a suitable value of *m* is found for the candidate. In this way a candidate's ID is suitable for encryption and remains consistent throughout the cryptographic process.

4.2.2 Shuffling Operations of the Mixes

Wang et al. propose the implementation of a re-encryption mix network for vote transmission. In their paper, the authors state that each mix should have a random permutation built-in to determine the relative input/output mapping of ports. In this scenario, a mix is considered secure as long as its permutation remains a secret; that is, as long as a single mix in the mix network remains secure, the security of the entire mix network is guaranteed.

Further, Wang proposes that several mixes in the network may be able to expose their permutation for the purposes of detecting a cheating mix. In this scenario, some mixes could be asked to randomly reveal half of their input/output mappings. This would allow a known message to be sent to a specific port on a mix. The output location of this message would then be known, and the re-encrypted message could be captured. If the captured message did not contain a re-encryption of the known message, this mix would be determined to have been dishonest (also known as a "cheating" mix) and this mix would then be replaced.

In this way we can detect a cheating mix with a probability of 50%. If multiple mixes were asked to reveal their input/output mappings, the probability of detection would become much higher.

In practice, we find no reason for a mix to have a consistent input/output mapping—it provides no benefit to use consistent mix port pairings, and randomizing the pairings adds a layer of anonymity for the voter. Further, having an inconsistent input/output mapping can help increase the honesty of the mix. When a mix is continuously randomizing input/output ports, it can be asked during testing to reveal 100% of its mixing operations through each of many rounds without exposing any detail an attacker could use.

Because we may know, for a specific round, the exact input/output pairings of a mix, it is thus possible to test the mix network as a whole for honesty. This would be done by submitting a known message ID to the first mix in the network. The entire network would then be asked to reveal its input/output pairings, which would allow us to follow the message through the mix network. The message could then be decrypted and checked at each stage, as well as in its final resting place on the Supervisor. If at any point the message ID were determined to have been tampered with, the offending device would be known and removed from operation.

For this reason our implementation uses a randomized input/output pairing for each round. At the end of each round, two arrays of ports (corresponding to inputs and outputs, respectively) are shuffled individually on each mix. Although a single shuffling operation would almost certainly be sufficient, these shuffling operations are based upon pseudo-random values generated using a microsecond seed (rather than the standard OpenSSL-generated random numbers) and this introduces a very slight possibility of a timing attack wherein the pairings could be predicted if the original order were known. The secondary shuffling operation adds an element of impossibility with regards to prediction of the output pairs.

If an operator wishes to be assured during testing that a mix is honest, the mix can be asked to generate a log file with all of its input/output pairings. We cover this improvement in more detail in 4.4.5.

4.2.3 Vote Injection Detection in the Mix Network

A concern arose early in the implementation of this thesis: what if an attacker were able to plug a device into the network that simply cast a vote for Candidate "C" repeatedly? If the mixes and the Receiver were to blindly accept, pass, and record every vote that was passed to it, it is conceivable that an attacker could capitalize on this by injecting as many votes as he or she felt necessary for his or her candidate to win.

This is worsened by the fact that the re-encryption network modifies the (c_1,c_2) value pair at each mix, making it impossible to determine whether a received vote is the result of a single legitimate ballot, or one of a thousand replays of an illegal vote (since all votes are indistinguishable until decrypted).

For this reason, we implemented a method of detecting vote injection. The method is simple: each port on the mix network and Receiver is able to receive one set of data on each port until all ports are full. Once all the ports are filled, the data is transferred to the next mix in the mix network. However, if a single port receives two sets of data, it assumes that someone is tampering with the network. To protect against this the mix writes the values it has already received to disk (to protect against lost votes), then shuts down. This creates a chain reaction which is shown in Figure 4.2.

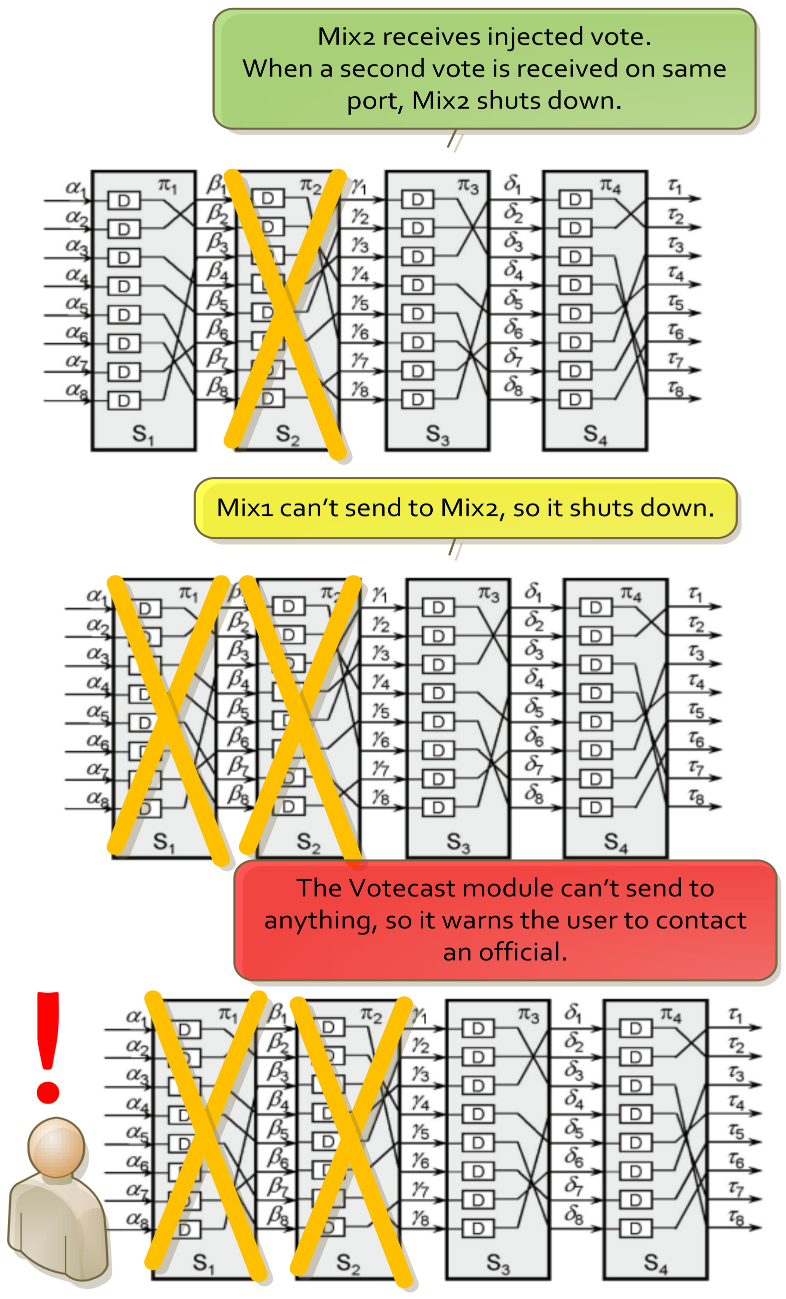

Mix2 receives injected vote. When a second vote is received on same port, Mix2 shuts down.
Mix1 can't send to Mix2, so it shuts down.
The Votecast module can't send to anything, so it warns the user to contact an official.
S1
S2
S3
S4

Figure 4.2: Cascading detection of a replay or injection attack

4.2.3.1 Advantages and Disadvantages of Vote Injection Detection

The condition created by a mix network that does injection detection is desirable in many respects. For one, it guarantees that each user can only cast a single vote because duplicate votes would trigger the same mechanism as an injected vote. It also guarantees that an attacker cannot artificially inflate the number of votes for a specific candidate without first capturing and deleting a valid vote (as in a man-in-the-middle attack).

On the other hand, vote injection detection creates conditions which may be difficult for poll monitors to maintain. For example, if detection is enabled then voters *must* maintain a strict round-robin pattern to the polling machines. If a voter is accidentally allowed to cast his or her vote on a machine that has already cast a vote during the current round, the duplicate vote would be detected as cheating and the network would need to be restarted. Suggestions for improving on this model are presented in Chapter 1.

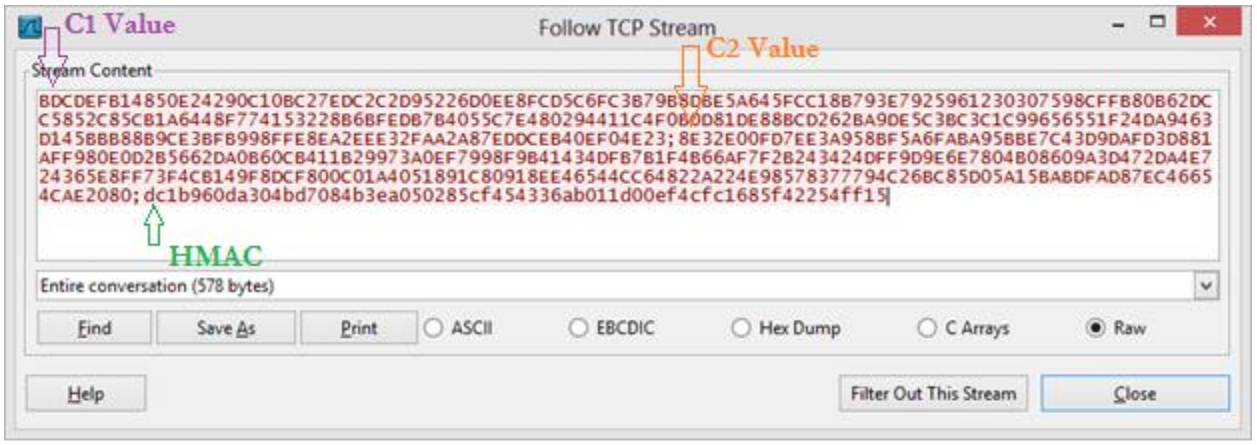
4.2.4 Message Authentication Hash Over (c_1;c_2) Pair

Re-encryption and decryption mix networks each have relative advantages and disadvantages, but neither is clearly the better solution. We can gain some of the advantages of a decryption network in a re-encryption network by hashing the message with a secret key to verify that the message has not been altered in transit.

A message authentication code (MAC) is a piece of information that can be used to authenticate the sender and guarantee the integrity of a message. A MAC also can be used to detect alterations to the original message—both intentional and accidental. By applying a hash-based message authentication code (HMAC) to the (c_1;c_2) key pair in our re-encryption mix network, we are able to create a hybrid re-encryption mix network with many of the advantages of a decryption mix network.

HMACs work by sharing a key between two hosts—in the case of our implementation each pair of adjacent mixes shares a secret key. The mixes then use these keys to hash the data sent and received (respectively) to determine if the data was modified in transit.

The key pair and its appended SHA-256 HMAC value can be seen in Figure 4.3 which is an actual packet capture from our mix network implementation. Key pair and HMAC values are separated using a semicolon delimiter.



```
BDCDEFB14850E24290C10BC27EDC2C2D95226D0EE8FCD5C6FC3B79B8DBE5A645FCC18B793E7925961230307598CFFB80B62DC
C5852C85CB1A6448F774153228B6BFEDB7B4055C7E480294411C4F0B0D81DE88BCD262BA9DE5C3BC3C1C99656551F24DA9463
D145BBB88B9CE3BFB998FFE8EA2EEE32FAA2A87EDDCEB40EF04E23; 8E32E00FD7EE3A958BF5A6FABA95BBE7C43D9DAFD3D881
AFF980E0D2B5662DA0B60CB411B29973A0EF7998F9B41434DFB7B1F4B66AF7F2B243424DFF9D9E6E7804B08609A3D472DA4E7
24365E8FF73F4CB149F8DCF800C01A4051891C80918EE46544CC64822A224E98578377794C26BC85D05A15BABDFAD87EC4665
4CAE2080; dc1b960da304bd7084b3ea050285cf454336ab011d00ef4cfc1685f42254ff15
```

Entire conversation (578 bytes)

( ) ASCII ( ) EBCDIC ( ) Hex Dump ( ) C Arrays (●) Raw

Figure 4.3 Packet Capture Showing C1;C2;HMAC Values

4.3 Non-Technical Workflow

This section documents the non-technical workflow used by our e-voting implementation. The workflow begins when a voter approaches a terminal, as shown in Figure 4.4.

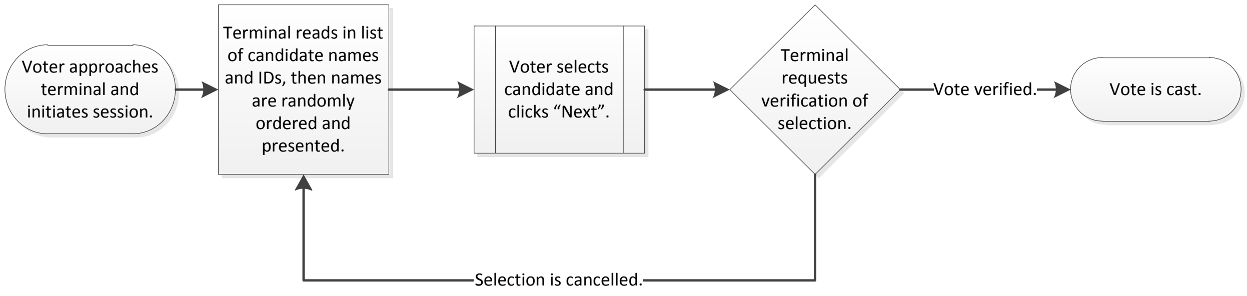

Voter approaches terminal and initiates session.
Terminal reads in list of candidate names and IDs, then names are randomly ordered and presented.
Voter selects candidate and clicks "Next".
Terminal requests verification of selection.
Vote verified.
Vote is cast.
Selection is cancelled.

Figure 4.4 Prototype Workflow (Non-Technical)

4.3.1 Terminals

Terminals (the graphical front-end to our e-voting solution) have several major components, which must follow a specific order of execution. The following sequence of events must occur in order, as shown in Figure 4.5.

4.3.1.1 *Candidate Values*

Candidate names are retrieved from the file */opt/evoting/candidates/candidates.txt*. This file also contains a random ID for each candidate. This process is implemented by the Prevoting module and full details on this operation are discussed in 4.4.1.

4.3.1.2 *Randomization of Candidate Names*

An ideal system presents candidates' names in a manner that prevents one candidate from having an advantage. For example, a candidate who is listed first alphabetically may have a perceived advantage. Thus, when a candidate's name is read from the database or text file, it is assigned a random value that is used to order the list of names.

A "true random" number is not necessary so we employ the combination of the C function *srand( )* seeded with a *time( )* variable to create a pseudo-random integer. Each integer is assigned to a candidate and determines the order of candidates' names.

4.3.1.3 *Verification of Candidate Selection*

Once a user selects a candidate and opts to proceed, we present the candidate selection for verification. During this verification the user has the option to cancel or proceed. This allows a user to correct a mistake or change his or her mind without losing their vote.

4.3.1.4 *Encryption and Transmission*

After the voter selects and verifies their candidate selection, the vote must be encrypted. The resulting transformation described as $\{c_1, c_2\}$ is printed to a receipt for the user and is sent to the first mix in the mix network, while simultaneously being appended to the bulletin board.

4.4 Technical Details: Prototype Implementation

This implementation contains several executable binaries, which we will now discuss in their prescribed order of use. All prototype files are intended to be run from the */opt/evoting/* directory, which we call the "evoting directory". A *Makefile* is included in this directory, allowing a user to link and compile all source files located in */opt/evoting/src/* into executable binaries by issuing the command '*make*' from the evoting directory. In order to compile the binaries, a machine needs the following libraries, available via Aptitude on Debian-based systems:

- OpenSSL
- Libssl-dev
- Libevent
- Libevent-dev

Further directions and dependencies can be found in the README file located in the root evoting directory.

The directory tree for */opt/evoting/* is:

- *src/* -- Contains the source code for all binaries.
    - *src/build/* -- The build directory for all binaries.
- *crypto/* -- Contains text files for all necessary cryptographic values.
- *candidates/* -- Contains two files:
    - *candidates.txt* – A listing of candidate names, 1 per line. If candidates have a number assigned to their name, this number appears after a semicolon (;) on the same line.
    - *randcand.txt* – A randomized listing of candidate names used by the voting terminal.
- *mix/* -- A folder containing the binaries for each mix in the mix network.
- *threshold/* -- A folder containing Python scripts for generating and recovering keys split using Shamir secret sharing. A README file provides greater documentation on use.
- *debug/*-- A folder containing debugging information, if debugging is enabled in the binaries. This folder also receives errors from the web service that presents the Terminals.

- *testing/* -- For purposes of testing this prototype, a number of binaries and scripts have been placed in this folder:
    - *startmixes.sh* – automatically STARTS all mixes as well as the R   eceiver.
    - *stopmixes.sh* – Automatically STOPS all mixes as well as the Receiver.
    - *onetest.sh* – Sends a single message to the first mix.
    - *fulltest.sh* – Automatically shows full functionality of the voting machine including encryption, transmission, and standard decryption.
    - *votecast[x]* – These pre-compiled binaries are programmed to send to port [x].
    - *injection.sh* – Simulates a vote injection attack, described later in this chapter.
- *votes/* -- The votes folder is where the Receiver writes its data, as well as the location for the bulletinboard.txt file.

We now move to discussion of the binaries in the order in which they are to be used. This entire process, including the workflow and binaries used, is documented in Figure 4.5.

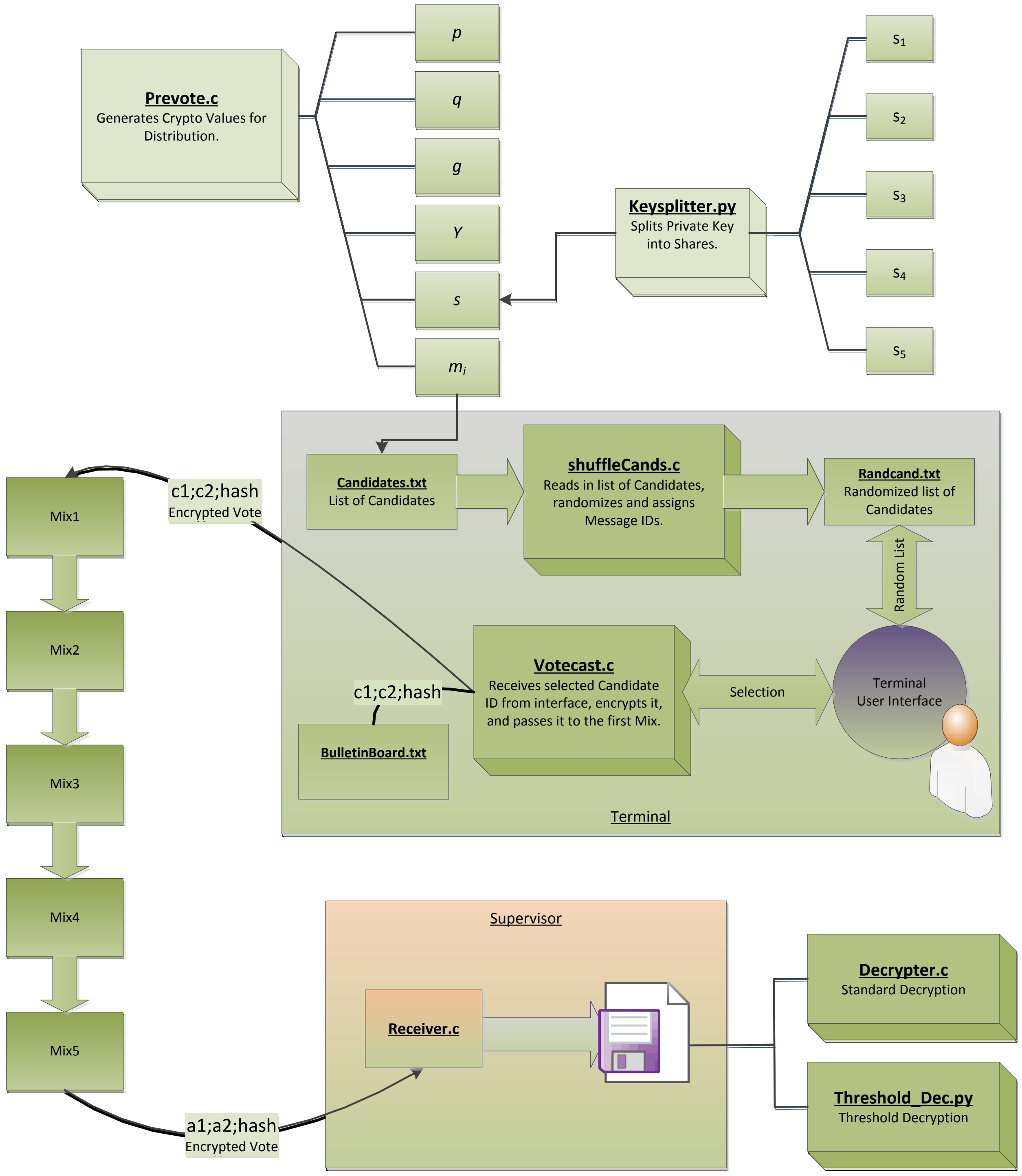


Figure 4.5 E-Voting Module Flow

4.4.1 Prevote

Before voting can begin, there are a number of variables that must exist on all voting machines in the precinct. These variables are *p, q, g,* and *Y*. The *s* value, or private key, is also required but must be stored in a protected area until needed for decryption.

The purpose of Prevote is to generate these variables as described in 4.1.2 and write them to files for distribution. These files should be placed in the */opt/evoting/crypto* folder, which Prevote does automatically. The file names expected are as follows:

- *p.txt* – the *p* prime.
- *q.txt* – the *q* prime.
- *g.txt* – the *g* value.
- *pub.key* – the supervisor's public key (*Y*).
- *priv.key* – the supervisor's private key (*s*).

Prevote reads in the list of candidate names and verifies that the candidate ID's are members of <g>. To test this, we determine whether $\mathbf{m^q \equiv 1 \bmod p}$. If the ID's (known as *m*) are not in <g>, then Prevote replaces the ID with one that is a member of <g>. If the candidates do not have ID's then Prevote generates them and appends them to *candidates.txt*. The *candidates.txt* file should then be placed on all terminals in the precinct.

Prevote contains several *#define* statements at the top of the source code which may be used to alter program behavior.

- *KEYSIZE* – Used to set the size of the key space. Defaults to 1024.
- *DEBUG* – If set to 1, Prevote creates more verbose output. Defaults to 0.

Prevote must be run before any other binaries if cryptographic values are not present already. The values found in the *crypto* folder are used by almost every other executable. The Prevote module is used once at the beginning of the election cycle and does not need to be present on voting terminals.

It is important to note that the *s* value, or the private key, should not be present on any machines except the Supervisors and/or a carefully guarded remote store. Like any private key, it must be protected or the security of the entire e-voting system is compromised.

4.4.2 Keysplitter

To implement threshold decryption as described in section 4.1.3, the python script Keysplitter.py is provided in the *threshold* folder. The Keysplitter.py script has several required flags that can be seen by supplying '-h' or '—help' as a command line argument:

```
usage: keysplitter.py [-h] -n [nVal] -t [tVal] --priv_key [priv_key]

Split a single private key into "n" keys, where only "t" is required to be
present for decryption.

optional arguments:
  -h, --help            Show this help message and exit
  -n [nVal]             Number of key shares for distribution
  -t [tVal]             Number of shares required for decryption
  --priv_key [priv_key]    Private key to split
```

All arguments to Keysplitter.py are required. When finished, Keysplitter.py outputs keys to a file named with their "part" value. This filename must be retained for later reassembly. For example, priv_key.part.3 is key pair <3, s_3>.

The Threshold_dec.py script, which is discussed later, is responsible for using these keys for re-assembly of the private key. Neither of these scripts needs to reside on a voting terminal, and in fact, may be run on a machine completely independent of the voting process.

4.4.3 ShuffleCands

The ShuffleCands module resides on each voting terminal. When a voting session is initiated via the web interface, the ShuffleCands module gets called to perform a shuffling operation on the values found in the *candidates.txt* file. The resulting random candidate listing is then written to *randcand.txt*, which is used by the terminal to display the list of candidates to the voter.

4.4.4 Votecast

The final stage in voting is the execution of the Votecast command. This module resides on the voting terminal and takes a single argument, the voter ID of the candidate selected.

```
./votecast [unique_id]
```

Upon receipt of a unique identifier (known as *m*), Votecast first verifies that the ID can be converted to a BIGNUM value to ensure that no errors have occurred. It then retrieves the stored cryptographic values from the *crypto/* folder and generates a random value *r* as described in section 4.1.2.

Next, Votecast encrypts the message into the struct *c* = (c_1, c_2) as described in section 4.1.2. The message is then serialized using the *serialize*( ) function. This function receives two members of a struct and combines them into a single character array in the format **c_1;c_2**. Finally, an HMAC value is created using the key defined in the macro statement HMAC_KEY in Votecast's header. The HMAC value is then appended to the key pair as **c_1;c_2;HMAC**.

Now the message is ready to send. Votecast utilizes a *#define* variable SENDPORT to determine which port to send to as well as SENDHOST which is set to the IP address of the first mix in the mix network. In a production environment, eight voting terminals exist, each with a unique SENDPORT hard-coded into the Votecast module.

Once Votecast has successfully completed, it return the *c* struct with the encrypted candidate's ID, which will be displayed by the terminal. At the same time the (c_1, c_2) pair is written to an append-only file named *bulletinboard.txt*. This unique encrypted ID can be used by the voter to verify that his or her vote was correctly published to the bulletin board.

4.4.5 Mix[x]

The *mix/* directory contains several binaries with names like Mix1, Mix2, etc. Each of these binaries represents a single mix, multiples of which can be combined to create a mix network as described in section 4.1.4. The mix uses the *eventlib2* library to poll each of its open ports for data. When data is received on one of these ports, it initiates a callback that directs the data received into an *evbuffer* (event buffer). This data is handled as follows:

1. The mix verifies that data does not already exist on that port. (If it does, there is a potential that an attacker is attempting to inject data—the mix shuts down and logs an error).
2. A data struct is initialized for the data.
3. The data is de-serialized into this struct.

4. An HMAC hash is made using the previous mix's key and compared to the HMAC appended to the values. If the value does not match, log an alert.
5. The data is re-encrypted using a random value *t* that is generated at the time of the function call.
6. The data is re-serialized into a message for transport.
7. An HMAC value is created for the newly transformed message using a key specific to the mix.
8. The message is placed into a queue to be sent to the next mix in the mix network via a random port. Alternately, if the next mix in the mix network is unavailable, the mix logs an error and exits.

Normal behavior for a mix network is to wait until data has been received on all ports before forwarding to the next mix in the net. This prevents timing-based attacks and ensures anonymity of the voter. For the purposes of this prototype, a #*define* variable TESTING is used to indicate whether the mix should wait for further data before sending. If TESTING is set to 1 (enabled), the mix simply re-encrypts the received data and forwards it directly to the next mix in the mix network. This is done to make testing of the entire system easier, without having to submit eight votes to see the system in operation. The TESTING mode also allows an Authority to test the mix network for honesty with a known message ID. During normal usage, the mix waits until more data is received.

This mix implements a Fisher-Yates shuffle [15] using a value generated by srand48( ) seeded by the microseconds of the current time, to shuffle the output of data as well as the port to which it is sent. This shuffle is applied to two arrays of digits 0-7, which are then applied to the output of the mix, as seen in the following pseudo code:

```
#define BASEPORT 32000
char sendArray[8] = [0, 1, 2, 3, 4, 5, 6, 7];
char portArray[8] = [0, 1, 2, 3, 4, 5, 6, 7];
shuffle(sendArray);
shuffle(portArray);
sendArray[8] = [7, 3, 0, 1, 2, 5, 4, 6] //New Values
portArray[8] = [3, 1, 7, 0, 4, 5, 2, 6] //New Values
for(int i=0; i<8; i++) {
        TCP_send msg[sendArray[i]] -> PORT(BASEPORT + portArray[i] + 1);
    }
```

Figure 4.6 Pseudo Code for Input/Output Shuffle

If logging is enabled on the mix shown above, a log entry like the one in Figure 4.7 is generated.

```
Mix2: Sending data[7] to [32004]
Mix2: Sending data[3] to [32002]
Mix2: Sending data[0] to [32008]
Mix2: Sending data[1] to [32001]
Mix2: Sending data[2] to [32005]
Mix2: Sending data[5] to [32006]
Mix2: Sending data[4] to [32003]
Mix2: Sending data[6] to [32007]
```

Figure 4.7 Log for Input/Output Shuffle

In this example, a random message is serialized and then sent to a random port within the next mix's range. The next time data is received by this mix, it shuffles both its inputs and outputs independently. For this reason, it is acceptable for a mix to reveal all of its input/output pairings for verification of mix honesty as discussed in section 4.2.2. It is not possible for an attacker to determine the pairings that occur with any accuracy. Figure 4.8 shows the port configuration as well as the re-encryption and forwarding mechanisms of the mixes.

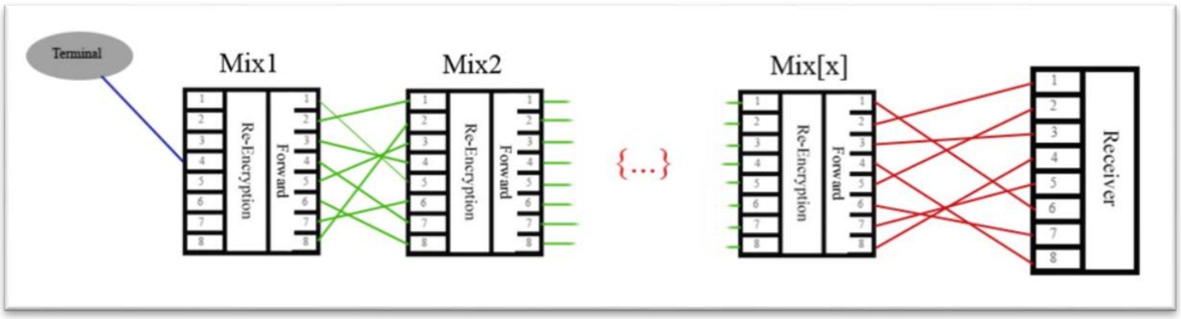


Figure 4.8: Mix Detail with Sample Input/Output Pairings

There are several important *#define* variables to be aware of in the Mix binaries:

- *LOGFILE* – The location of the output logging file. Logging is for testing only and is disabled on a production machine because it identifies the sender of a message.
- *MIXPORT* – The base port from which the mix derives open ports (+[1-8]).
- *MIXNUM* – The position of the mix in the mix network. Used only for logging.

- *NEXTMIX* – The base port of the next mix in the net. Used for forwarding.
- *NEXTHOST* – The IP address of the next host in the mix. Defaults to "localhost".
- *DEBUG* – Used to enable (1) or disable (0) verbose program output.
- *LOG* – Writes logging values to an output file in *debug/*. Enabled logging may lead to discovery of the vote source. Do NOT use in production.
- *TESTING* – Causes the mix to pass data without waiting for data to be received on all eight ports. Enabled (1) or disabled (0).
- *MY_HMAC_KEY* – A key unique to the mix. Used to create HMAC values for outgoing messages.
- *PREV_HMAC_KEY* – The key of the previous mix, used for comparisons on incoming messages.

Mixes are started by executing the module with an ampersand (&) symbol at the end of the command line to background their functionality (e.g., */mix1* &). Alternately, the script startmixes.sh in the *testing/* folder can be run to start all five mixes and the Receiver. Each mix tracks its process ID and can be stopped using the stopmixes.sh script, also located in the */testing* folder.

This prototype makes use of 5 mixes, however the mix framework is designed to be completely flexible and thus can be scaled nearly indefinitely. A chain of several mixes or several hundred mixes is completely possible within this framework.

4.4.6 Receiver

The Receiver is a simplified mix. Rather than receiving data and performing transformations on it, the Receiver simply receives the data, verifies the HMAC value, and writes the $c_1;c_2$ pair to disk. The Receiver module is run on the Supervisor. The default output location is */opt/evoting/votes/votes.txt*, but the output location can be changed using the *#define* value *VOTEOUTPUT*.

4.4.7 Threshold_Dec & Decrypter

Now that data has been generated, sent through the mix network, received, and written to disk, the final step is to decrypt the votes received. The Threshold_dec.py module handles threshold decryption as described in section 4.1.3, while Decrypter handles standard decryption as described in section 4.1.2. Threshold_dec.py does not perform any actual decryption. Rather, it takes in a number of "key part" files as arguments and returns the recovered key. This key can then be supplied to Decrypter for final decryption. The optional '–out' flag of Threshold_dec.py specifies that the recovered key should be written to the specified file. Usage is as follows:

```
usage: threshold_dec.py [-h] [--out OUTFILE] file [file ...]

Combine split keys to recover a private key.

positional arguments:
  file          key files to combine (include multiple!)

optional arguments:
  -h, --help     show this help message and exit
  --out OUTFILE  write the decoded key to a file
```

Decrypter takes in a private key, *s*, to use for decryption. By default Decrypter is configured to use the "priv_key.txt" located in the *crypto* folder. Decrypter prints the decrypted votes to screen, allowing the user to redirect the output in any number of ways using standard Linux pipes.

4.4.8 Voting Interface

The voting interface is coded in PHP with system calls made to the binaries described above. The output is then captured and formatted for display. For this prototype, emphasis was not placed on making the voting interface fully compliant with the Americans with Disabilities Act. No options are provided for handicap access or alternate languages.

The interface begins with a banner displaying the county and provides space for directions and/or options for user assistance. Upon clicking "Next" the user is redirected to *select.php* where a list of randomized candidates and their party affiliations is presented with a radio button next to the candidate names. By the nature of the radio button only one selection at a time is allowed, thus preventing over voting. When the user is ready to proceed with their selection he or she can choose "Next" to continue (the use of the word "next" also implies that the user is not finalizing the vote).

The next page, *confirm.php,* re-iterates the candidate selection by name and allows the user to either "Cancel" (return to the candidate selection page) or "Vote". If the voter selects "Vote", the candidate's unique ID is submitted to Votecast on the operating system, which returns a unique encrypted value pair representing the voter's choice. This value is then printed to paper to allow the voter to retain confirmation of the vote; it is also appended to the bulletin board for later verification.

All code for the interface is contained in */var/www* on the host operating system and is only accessible and modifiable by the *root* user. To be run on the host operating system, php5 and apache2 are required.

Prior to being installed on a system, the web pages that comprise the terminal are located in */opt/evoting/webterminal.* A script named *deployweb.sh* is located in this directory and can be used to automatically deploy the terminal after the necessary libraries have been installed.

4.4.9 Tallying Votes

The standard Decrypter module is designed to decrypt votes and print them to standard output (*stdout*). This means that if the Decrypter module is run without any modifications, the votes will simply be printed to the screen. This is done to allow a wide variety of possible tallying methods. A few potential methods follow:

- Output of decrypted votes to a text file: Using a standard Linux *redirect* the decrypted votes could be placed into a new text file. For example:

  ```
  ./decrypter > decrypted_votes.txt
  ```

  This would allow the votes to be tallied at another time, or for the raw unencrypted votes to be transferred to a central tallying station.

- Command-line tallying: Using Linux pipes and the standard programs *sort* and *uniq*, the votes could be tallied on the fly. For example:

  ```
  ./decrypter | sort | uniq –c
  ```

  This returns the candidate ID with corresponding tallies to the screen. Using a known-good (i.e. hashed) version of *sort* and *uniq* would produce trustable results.

Due to a nearly infinite number of possibilities, the actual tallying method used is left to the Terminal Authorities.

Chapter 5 Security of the Prototype

The potential for an attack on a system such as this is extremely high, and analysis would not be complete without a review of the system from an attacker's perspective. In this section, we analyze a number of potential attacks on the system and their effectiveness.

5.1.1 Network Traffic Analysis

We begin attacking our system by capturing network traffic on the loopback interface. Note that in a production environment, each mix exists as a separate device on the network, meaning that an attacker only has visibility into traffic between two hosts. However, in our prototype system all network traffic is visible because it is all run from a single machine.

Because the local packet capture gives us access to all communications on the prototype, we can see that the timing of the vote can be determined if an attacker is able to gain access to the network between the terminal and the first mix. Importantly, the contents of the votes during transmission are securely encrypted and indistinguishable. Thus an attacker knows which $c_1;c_2$ value pair belongs to which voter, but not the candidate that the voter has chosen.

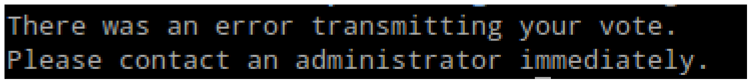
Packet streams found later in the packet capture as the votes traverse the mix network are completely indistinguishable. These streams are what an attacker would see if they successfully inserted themselves between two mixes, or between the last mix and the Receiver. The streams originate at OS-assigned sending ports and go to mix ports in an out-of-order sequence (e.g., 31003, then 31006, etc.). It is therefore impossible to determine the origin of these votes or their content.

This packet capture shows the security we desire at the network level and exactly what an attacker would hope against: the data flowing through the network is nothing but random-appearing encrypted data which lacks attribution and information.

5.1.2 Replay/Injection Attacks

What if an attacker knew the contents of a vote, and wished to replay this vote continuously? For example, if an attacker placed a vote for "Bob Dole", and compromised a section of the network, could the attacker continuously broadcast his vote to multiply the number of votes for that candidate without interrupting the voting process?

To test this attack, a single vote was injected into the second port of the first mix (port 31002). Eight votes were then transmitted in the correct sequence to the first mix. The following output was observed as the votes were being sent to port 31002 on the first mix:



```
There was an error transmitting your vote.
Please contact an administrator immediately.
```

Figure 5.1: Error generated by vote injection

This error would be shown on-screen to a voter. This behavior is by design, because the mixes were made to protect against these kinds of attacks: when the mix reads data in on one port, it first verifies that data has not already been received on this port during this round. If there is already data present, the mix is configured to log an error and shut down. The upstream mix is then unable to send, and also logs an error and shuts down. Finally, the Votecast module is unable to reach any mixes, which causes it to display an error to the user indicating that an official needs to be contacted. Figure 5.2 shows the exact logic used by a mix when sending and receiving data. The reverse-cascade effect of an injected vote is illustrated in and was previously discussed in section 4.2.3.

The *testing* folder in our implementation contains a bash script "injection.sh" which automatically simulates a vote injection, complete with verbose output and log display.

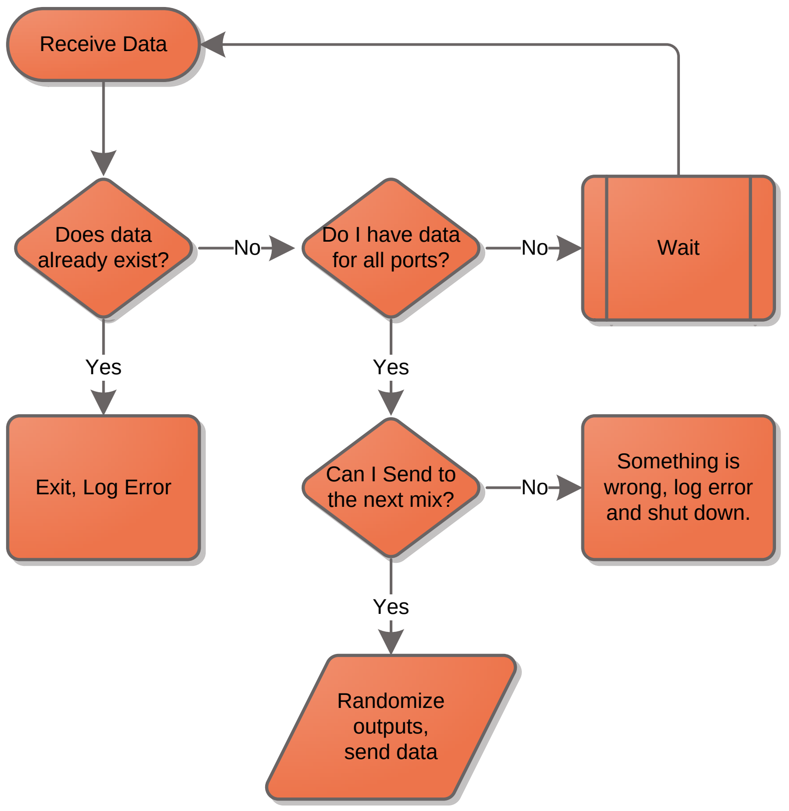

Receive Data
Does data already exist?
No
Do I have data for all ports?
No
Wait
Yes
Yes
Exit, Log Error
Can I Send to the next mix?
No
Something is wrong, log error and shut down.
Yes
Randomize outputs, send data

Figure 5.2 Mix Logic

5.1.3 Man-in-the-Middle Attacks

What if an attacker places him or herself between two mixes, then intercepts and alters or drops the messages? This type of attack, called a "man-in-the-middle" (MITM) attack, could be employed if networks were used in which an attacker could insert themselves on the same network as one of the mixes. Note that this type of attack is similar to the injection attack above, but requires the additional step of discarding the votes sent by the previous mix so that the next mix never receives them.

This type of attack would not be detected by the mixes as an injection because a duplicate vote would not be received. However, the mix network present in this implementation makes use of an additional security mechanism which makes this kind of attack very difficult to carry out successfully.

The hashed message authentication code discussed in 4.2.4 requires any attacker who is attempting to successfully carry out a MITM attack to know the secret key shared between the two mixes he or she is injecting between. Any attempt that uses an incorrect key results in a red flag being raised in the logging mechanism of the receiving mix.

For this reason, it is extremely unlikely that an attacker can successfully hijack network traffic without first gaining privileged shell access to one of the mixes. In the extreme case that an attacker hijacked a mix, they could potentially be able to cast unlimited votes for a candidate but they would still be unable to decrypt the votes arriving at the mix.

5.1.4 Timing Attacks

All random values used for cryptographic operations are based on the OpenSSL BN_rand functions. These values are a major step above a standard pseudo-random number generator (PRNG) and are considered to be random enough for encryption use. However, these values lack the full spectrum of entropy found with a true random number generator—OpenSSL bills BN_rand as a "cryptographically strong pseudo-random number generator". An academic literature search turns up no relevant theoretical attacks against this library, and we have used the latest version available which has no known bugs.

Several algorithms such as the port and message shuffling done on the mixes are done using microsecond precision random values generated by the system. Although these high-precision random values should be sufficient on their own, these shuffling operations are also double-random (one random operation for the ports, one random operation for the messages). Since each system is not publicly networked, attacks would have to be carried out in person. This introduces an element of impossible timing, and far too many variables are present for an attacker to be able to mount a precise attack.

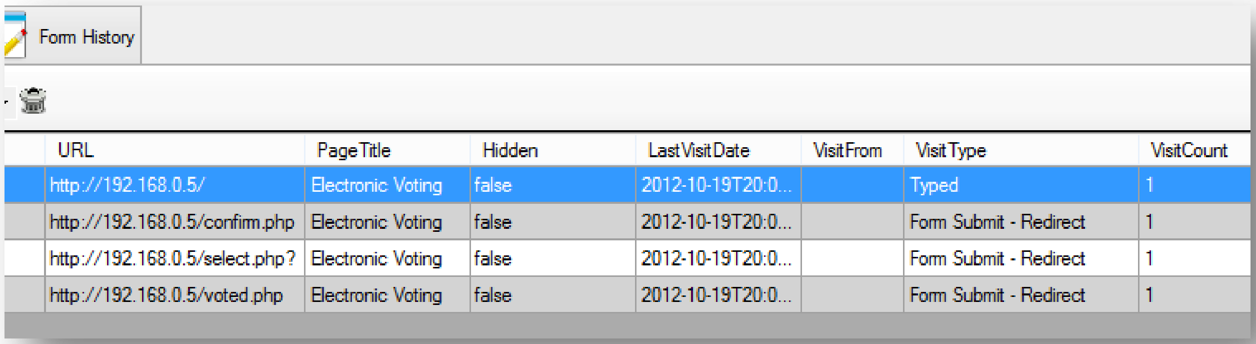
If an attacker were able to gain privileged access to a voting machine or mix's underlying operating system, it is conceivable that they would then be able to gather enough information to make a timing attack possible, increasing their chances of predicting the random number generated and determining with potentially high likelihood which candidate was chosen. For this reason and many others we recommend that all physical machines be carefully protected against both physical attacks (by closing off all USB and similar ports, for example) and network-based attacks (via open ports and unnecessary services).

5.1.5 Internet History Attack

The use of the web browser as an interface to the system provides a number of benefits with regards to terminal display, text sizing, and aid to users with disabilities--but it also creates a messy situation which can be forensically analyzed and potentially exploited by an attacker.

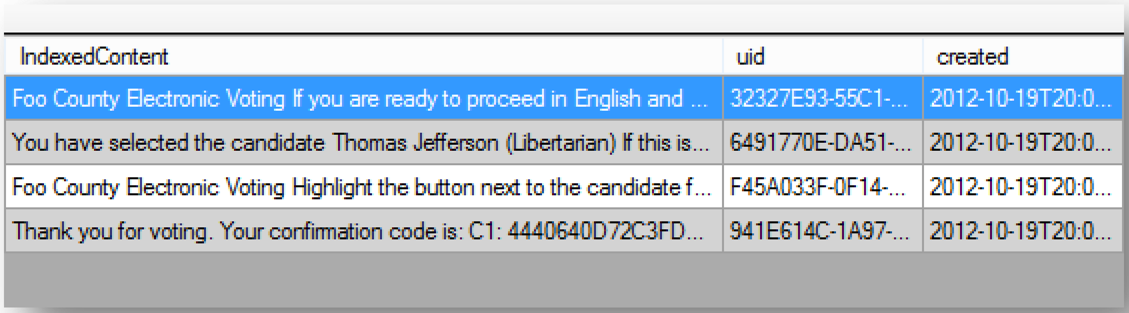
The problem is created as a result of the web browser "caching", or saving small snippets of data to allow faster rendering of the page. To create an internet history attack, we visited the Terminal site using the Firefox Browser. A vote was placed, and a confirmation number was received.

A number of tools are available for web history forensics, but for this attack the web cache was analyzed using Mandiant Web Historian [19]. The results were worse than expected: not only is the full history available, but it included time stamps and the complete contents of the voting page. Figure 5.3 and Figure 5.4 contain data retrieved by Mandiant Web Historian.



URL	PageTitle	Hidden	LastVisitDate	VisitFrom	VisitType	VisitCount
http://192.168.0.5/	Electronic Voting	false	2012-10-19T20:0...		Typed	1
http://192.168.0.5/confirm.php	Electronic Voting	false	2012-10-19T20:0...		Form Submit - Redirect	1
http://192.168.0.5/select.php?	Electronic Voting	false	2012-10-19T20:0...		Form Submit - Redirect	1
http://192.168.0.5/voted.php	Electronic Voting	false	2012-10-19T20:0...		Form Submit - Redirect	1

Figure 5.3: History of Pages Visited



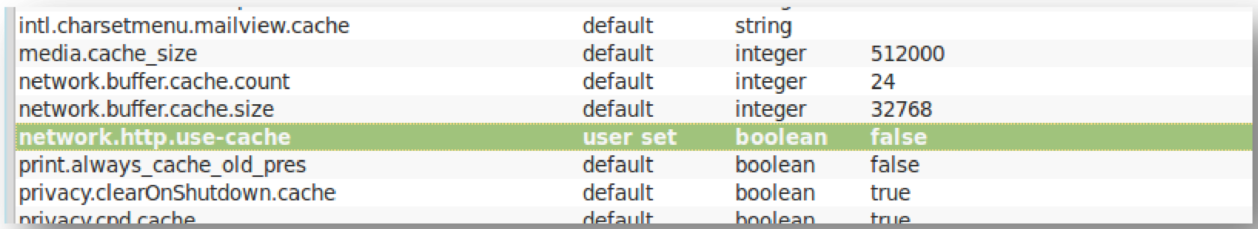
IndexedContent	uid	created
Foo County Electronic Voting If you are ready to proceed in English and ...	32327E93-55C1-...	2012-10-19T20:0...
You have selected the candidate Thomas Jefferson (Libertarian) If this is...	6491770E-DA51-...	2012-10-19T20:0...
Foo County Electronic Voting Highlight the button next to the candidate f...	F45A033F-0F14-...	2012-10-19T20:0...
Thank you for voting. Your confirmation code is: C1: 4440640D72C3FD...	941E614C-1A97-...	2012-10-19T20:0...

Figure 5.4: Contents of the web page

As seen above, the plaintext candidate name as well as the complete confirmation code is stored in the cache, leading to a full compromise of the vote integrity.

An easy solution to the problem above is to completely disable disk caching in whatever browser is being used. This forces the browser to use memory for temporary internet caching, which is both volatile (meaning it disappears if the system loses power) and should be encrypted according to our e-voting solution requirements. Alternately, the browsers memory location could be zeroed-out at the end of each voting session.

To correct this problem, we disabled caching on Firefox by entering "about:config" into the URL bar, searched for "cache", and then double-clicked *network.http.use-cache* to set the value to **false**.



```
intl.charsetmenu.mailview.cache        default     string
media.cache_size                       default     integer    512000
network.buffer.cache.count             default     integer    24
network.buffer.cache.size              default     integer    32768
network.http.use-cache                 user set    boolean    false
print.always_cache_old_pres            default     boolean    false
privacy.clearOnShutdown.cache          default     boolean    true
privacy.cpd.cache                      default     boolean    true
```

Figure 5.5: A secure Firefox Configuration

After making these changes we cleared Firefox's cache, reloaded the browser, and visited the site again. No cache data was present when the internet history was viewed again using Web Historian.

5.1.6 Binary Strings Analysis

A major problem with compiling a secret key such as our HMAC key into a binary is that someone may reverse-engineer the key out of the binary. In the case of our implementation, it turned out to be even simpler: the linux 'strings' command found our HMAC key almost immediately as seen in Figure 5.6.

```
ccollord@mint13vm /opt/evoting $ strings votecast
/lib/ld-linux.so.2
__gmon_start__
_Jv_RegisterClasses
libcrypto.so.1.0.0
BN_new
BN_clear
BN_bn2hex
libc.so.6
Err: fileread fopen
Value copy error
e07ff12afb57f661a32a49abb95ea36a
%02x
%s;%s
/opt/evoting/crypto/pub.key
Err: rand_range
FATAL ERROR: 'g' value is not a member of <g>.  Re-run 'prevote' binary.  Exiting.
Err: calc c2 mod_mul
Socket failed.
;*2$"
```

Figure 5.6 Truncated Strings Output

As easy as it is for an attacker to extract the HMAC key since it is stored in plaintext, it is of relatively little importance. The *mix* binaries are intended to be compiled and deployed with a new and unique key at each location. This means that the specific mix module and its unique password only exists on the machine hosting the mix—so to extract the HMAC key, the attacker must compromise the machine hosting the mix. In the case of an event such as this the attacker must guess the correct HMAC for the next mix, at the risk of an alert if the wrong key is chosen. Once again the need for strong physical and network security is highlighted and we do not find this to be a fault of the mix network.

Chapter 6 Summary & Direction for Future Study

There is no perfect electronic voting solution. The devices presented by academia place a heavy focus on security and redundancy while devices presented by industry place more emphasis on aesthetics and mobility—features that likely do more for sales than for voters. That is not to imply that one set of features is more important than another, because the most secure cryptography in the world does nothing if the average person cannot figure out how to use it.

Installation and maintenance of the base operating system and libraries upon which our solution is built requires specialized knowledge that cannot be expected from an average poll administrator. To be truly viable, this implementation needs to be rolled out as a single installation with thorough testing of both the OS and electronic voting utilities. Such an OS should be gutted of all unnecessary functionality and receive a thorough source review. TinyOS [20] is one solution that could be easily run on commodity hardware.

6.1 Areas of Improvement

Our implementation lacks some features identified as requirements for an ideal e-voting solution. The following areas need continued development before our prototype can be considered ideal:

- **[10] Compliance with the Americans with Disabilities Act (ADA):** Our implementation lacks features such as large text, braille, and audio assistance for those with impairments. Some functionality such as text enlargement already exists in most browsers. Internet to braille converters (known as "Web-Braille") already exist as well. These enhancements are not difficult to incorporate, but are beyond the scope of the prototype.
- **[12] Full Encryption of the Ballot:** Although the ballot is fully encrypted as required, the machine on which the applications are run is not fully encrypted. An adversary with access to the machine could easily recover internet history from memory even if disk caching was disabled, and discover how a user had voted. Cryptkeeper [21] provides a potential solution for encrypted memory that shows a promisingly low overhead, and hardware encrypted hard drives are already commonly available.

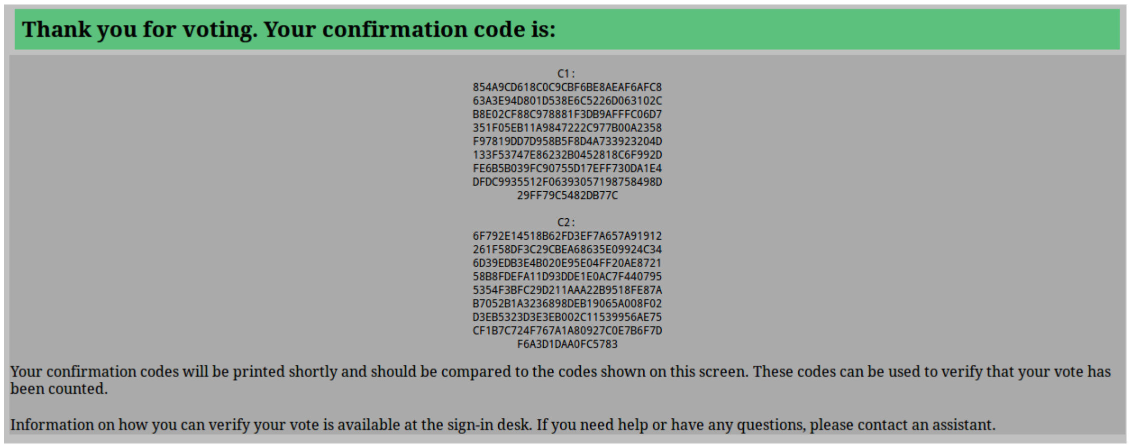
Another potential solution to this problem is to track memory addresses used during a voting session and clear this address space (by writing NULL values into this space) before the terminal is used again.

- **Round Robin Terminal Assignment:** As discussed, vote injection detection done by the mixes may cause problems for poll administrators. If a voter accidentally casts his or her vote on a machine in the incorrect order, it is detected as abuse and the mix network must be restarted. A possible solution to this problem is to provide a system whose sole purpose is to match a user at a terminal with an "open" voting session. This process should be transparent to the voter and is necessary to prevent problems.

  Another solution would be to allow the votecast module send to a random port on the first mix. This would require the first mix to be modified to allow for unlimited queuing of events on each port. Then, as full sets of eight votes are eventually realized on the first mix, it can begin its standard behavior of flushing data to the second mix. This would have the added benefit of providing more anonymity to the user on the first mix.
- **Fully Automatic Setup:** The mixes as well as the Votecast and Receiver binaries are relatively simple to set up for someone who is familiar with the C programming language--all important settings are found in the header of the respective source code under a #define macro and very few of these setting require changes. Ultimately, however, a poll administrator cannot be expected to understand this information nor be completely entrusted to make the changes needed for secure operation. A single executable, run at the time of deployment, should be included to generate secure HMAC keys, disable logging and debugging, and perform a full hash verification of the resulting binaries and other important system files.
- **Bulletin-Board Improvements:** As this implementation currently stands, if an attacker were to gain access to the bulletin board *and* the private key for decryption, a voter's privacy might be compromised. This is particularly true if the attacker knows the precise order in which a voter used the terminal. We see several potential improvements that could protect against this:
  - The bulletin board could be randomized using a technique similar to what the mix network uses, minus re-encryption.

  - The use of a decryption mix network rather than a re-encryption mix network would allow the voter's ballot to remain unchanged. Thus, the ballot could be checked on the Supervisor and the bulletin board could be eliminated.
  - Similar to above, the use of re-encryption on the currently-implemented mix network could be eliminated. In this way the key values returned to the voter could be checked against the values recorded on the Supervisor. The voter's identity would still be concealed by the shuffling actions of the mix network, and the key values would still be protected from modification by the HMAC value.
- **Vote Re-injection:** In some cases it may be necessary for an Authority to submit a vote to the system outside of the normal use-case. This solution has not been provided in this implementation. We see two potential reasons for this to occur:
  - **If a mix shuts down due to vote injection**: In this case, the votes currently held on the mix will be logged for later recovery. To prevent these votes from going uncounted, the Authority will need a method to re-submit these votes.
  - **Before closing the polls**: The mixes will only flush once they have received eight votes. This can cause an issue if the number of people voting modulus eight is not zero. For example: if twelve people voted, the mixes would submit eight votes to the Supervisor but the mixes would be left holding four votes indefinitely. A mechanism to submit null votes (valid messages not associated with a candidate) is needed to flush out these final four votes.

  However, it is important to note that this mechanism could provide a way for an attacker or a dishonest Authority to submit votes, and must be guarded appropriately.
- **Design Challenges:** Finally, Wang notes that he believes there are problems inherent to the design [1]. The authors note that it may be difficult for a voter to verify two values and to remember parts of these values when later comparing them with their printed receipt. In our implementation we attempted to improve upon this by formatting the number like a word seek puzzle. This allows a voter to verify several portions of the number quickly (for example, every 30th and 31st character—the beginnings and ends of every line).



Thank you for voting. Your confirmation code is:

```
                 C1:
854A9CD618C0C9CBF6BE8AEAF6AFC8
63A3E94D801D538E6C5226D063102C
B8E02CF88C978881F3DB9AFFFC06D7
351F05EB11A9847222C977B00A2358
F97819DD7D958B5F8D4A733923204D
133F53747E86232B0452818C6F992D
FE6B5B039FC90755D17EFF730DA1E4
DFDC9935512F06393057198758498D
       29FF79C5482DB77C

                 C2:
6F792E14518B62FD3EF7A657A91912
261F58DF3C29CBEA68635E09924C34
6D39EDB3E4B020E95E04FF20AE8721
58B8FDEFA11D93DDE1E0AC7F440795
5354F3BFC29D211AAA22B9518FE87A
B7052B1A3236898DEB19065A008F02
D3EB5323D3E3EB002C11539956AE75
CF1B7C724F767A1A80927C0E7B6F7D
       F6A3D1DAA0FC5783
```

Your confirmation codes will be printed shortly and should be compared to the codes shown on this screen. These codes can be used to verify that your vote has been counted.

Information on how you can verify your vote is available at the sign-in desk. If you need help or have any questions, please contact an assistant.

Figure 6.1: A sample vote confirmation.

6.2 Summary

Ballot technology is likely to remain an important topic for some time to come. We believe that it is critically important that a viable solution receives a full academic review and testing for production as soon as possible. In the 2012 election year, a small scandal broke out because Hart Intercivic, which produces eSlate, has been tied to Bain Capital. Bain Capital was previously owned by 2012 Presidential candidate Mitt Romney, so this appeared to be a conflict of interest [22]. Whether or not this truly was a conflict of interest, it highlights the importance of a fair vote tallying system and of transparency in the voting process. As Josef Stalin was famously quoted by his former secretary, "I consider it completely unimportant who in the party will vote, or how; but what is extraordinarily important is this — who will count the votes, and how."

The implementation presented by this thesis provides a cryptographically strong and open-source solution that is highly extensible and contains a number of advanced features. With further polishing and review we believe it can be an excellent candidate for use during an actual election cycle.

Appendix

In the following example, we detail the creation of cryptographic values using small prime numbers for demonstration. In our implementation, *p* is determined first, and *q* is derived. For this demonstration, we reverse the order for simplicity.

- **$q = 11$**
- **$p = 2q + 1 = 23$**
- g must be of order q, meaning: $g^q \bmod p = g^{2q} \bmod p = g^{3q} \bmod p = \ldots = 1$
    - Generate a random h and raise it to $(p-1)/q$, mod p:
        - Random: $h = 12$
        - $h^2 = 12^2 = 144 \bmod 23 =$ **((g = 6))**
        - Test: $g^q \bmod p = 6^{11} \bmod 23 = 1$ (success!)
- $\langle g \rangle = \{ g, g^2 \bmod 23, g^3 \bmod 23, g^4 \bmod 23, g^5 \bmod 23, g^6 \bmod 23, g^7 \bmod 23, g^8 \bmod 23, g^9 \bmod 23, g^{10} \bmod 23, g^{11} \bmod 23 \} = \{ 6, 13, 9, 8, 2, 12, 3, 18, 16, 4, 1 \}$

We now wish to encrypt a candidate "George Washington" who has an ID of 5. This candidate ID is not found in <*g*>, so we encrypt the additive inverse modulo *p*, or (23 – 5 =) **18**. As can be seen above, 18 is found in <*g*>. In our modification, several candidate IDs would be generated until a viable value was found.

References

[1] Wang, X., Grove, R. and Heydari, H.M. Secure Electronic Voting with Cryptography. In *Applied Cryptography for Cyber Security and Defense: Information Encryption and Cyphering*. IGI Global, 2010.

[2] CONGRESSIONAL DIGEST. *Directory Recording Electronic Voting Machines: Questions about Security and Reliability*. 262-288, 2006.

[3] Chaum, David. Secret-Ballot Receipts: True Voter-Verifiable Elections. *IEEE Security and Privacy Magazine* (2004), 38-47.

[4] Larocca, R. and Klemanski, J.S. U.S. State Election Reform and Turnout in Presidential Elections. *State Politics & Policy Quarterly* (2011), 76-101.

[5] Bohli, J.M., Henrich, C., et al. Enhancing Electronic Voting Machines on the Example of Bingo Voting. *IEEE Transactions of Information Forensics and Security* (2009), 745-750.

[6] Chaum, D. and Essex, A. et al. Scantegrity: End-to-End Voter-Verifiable Optical-Scan Voting. *IEEE Security and Privacy*, 6, 3 (2008), 40-46.

[7] Fan, C. I. & Sun, W. Z. *An Efficient Multi-Receipt Mechanism for Uncoercible Anonymous Electronic Voting.* Department of Computer Science and Engineering, National Sun Yat-sen University, Taiwan, 2008.

[8] Wolfinger, R.E. Voter Turnout. *Society* (1991), 23-26.

[9] Saltman, R.G. *Effective Use of Computing Technology in Vote-Tallying*. Washington, D.C., 1975.

[10] Harris, M. A Farewell to Hanging Chads. *Engineering and Technology* (2010), 23-25.

[11] Reynolds, A. and Steenbergen, M. How the World Votes: The Political Consequences of Ballot Design, Innovation, and Manipulation. *Electoral Studies #25* (2006), 570-598.

[12] Goldwasser, Shafi & Micali, Silvio. Probabilistic Encryption. *Journal of Computer and System Sciences* (1984), 270-299.

[13] Paillier, Pascal & Pointcheval, David. Efficient Public-Key Cryptosystems Provably Secure Against Active Adversaries. *Advances in Cryptology - Proceedings of ASIACRYPT '99* (1999), 165-179.

[14] Gonsalves, Chris. Will the Voting Machines Get it Right? *Baseline Magazine* (February 2008), 15-18.

[15] Technology, New Jersey Institute of. http://www.state.nj.us/state/elections/voter-critertia/vvpr-hearing-reports-06-07-08/ES&S_Final_Report_Sept%2026_2007.pdf

[16] Sandler, Daniel, Derr, Kyle, and Wallach, Dan S. VoteBox: A Tamper-Evident, Verifiable Electronic Voting System. http://static.usenix.org/events/sec08/tech/full_papers/sandler/sandler_html/index.html

[17] Publication, Federal Information Processing Standards. Digital Signature Standar (DSS). *FIPS PUB 186-3* (June 2009), 41.

[18] Frankel, Yair and Desmedt, Yvo. Threshold Cryptosystems. *CRYPTO '89 Proceedings on Advances in cryptology*

(1989), 307 - 315.

[19] MANDIANT. Mandiant Web Historian. http://www.mandiant.com/resources/download/web-historian

[20] TinyOS. http://www.tinyos.net/

[21] Peterson, Peter A. H. Cryptkeeper: Improving Security With Encrypted RAM. *Technologies for Homeland Security (HST)* (November 2010), 120-126.

[22] Bello, Gerry & Fitrakis, Bob. Vote counting company tied to Romney. http://www.freepress.org/departments/display/19/2012/4725

[23] Jakobsson, M., Juels, A., and Rivest, R.L. Making Mix Nets Robust for Electronic Voting by Randomizing Partial Checking. In *USENIX Security '02* (2002), 339-353.

[24] McCarthy, J. Machine Error Gives Bush 3,893 Extra Votes in Ohio. *USA Today* (November 6, 2004).

[25] McDonald, M.P. *United States Elections Project*. 2010.

[26] Technology, National Institute of Standards and. Fisher-Yates shuffle. http://xlinux.nist.gov/dads//HTML/fisherYatesShuffle.html