

UNIVERSITY OF WAIKATO

Mobile Voting (M-Voting) for Democracies of the Future

Author:

Brandon NICHOLSON

Supervisors:

Dr. Raja Naeem AKRAM

Dr. Ryan KO

*A report submitted in fulfilment of the requirements
for the degree of Bachelor of Computing and Mathematical Science (Honours)*

in the

Cyber Security Lab
Computer Science Department

October 2014

Abstract

For most of the world, democratic elections are currently conducted via paper ballots. In some places, paper ballots have been replaced by electronic voting booths. More recently, the idea of remote e-voting has surfaced. That is, using the Internet to vote remotely (such as from your own home) using your computer. The aim of this project is to assess the feasibility of developing a secure mobile application that allows voters to use their smart-phones to vote from wherever they happen to be at the time of the election. The main focus of the project is to make it secure, auditable and trustworthy to voters. The motivation behind shifting to a mobile platform is to increase voter turnout on election day, and make it easier for everyone to cast their vote without having to travel to a polling station.

Contents

Abstract	i
Contents	ii
List of Figures	v
List of Tables	vi
1 Introduction	1
1.1 Motivations	1
1.2 Project Aims and Scope	2
1.3 My Contributions	2
2 Background	3
2.1 Paper Voting	4
2.1.1 On-premises	4
2.1.2 Remote	4
2.2 On-premises E-Voting	4
2.2.1 Trusted Central Authority	4
2.2.2 Self-tallying	4
2.3 Remote E-Voting	5
2.4 Related Work	6
2.4.1 Self-enforcing Electronic Voting	6
2.4.2 Voting System Requirements	6
3 The Concept of Self-tallying	8
3.1 Terminology	8
3.1.1 Pseudo-Random Numbers	8
3.1.2 Public Keys	8
3.1.3 Finite Cyclic Groups & DDH Intractability	8
3.2 Ballot Generation	9
3.3 Cancellation Formula	10
3.4 Self-tallying Formula	10
3.5 Properties of a Self-enforcing Electronic Voting System	11
4 Proposed Architecture	12
4.1 The Role of Human Authorities	12
4.2 Architecture	13

4.2.1	Architecture Overview	13
4.2.2	Authentication Procedures	13
4.2.3	Voting Procedures	14
4.3	Challenges: Coercion	15
5	Implementation	16
5.1	Proof of Concept Overview	16
5.2	Implementation	16
5.2.1	M-Vote Application	16
5.2.2	Creating the Servers	17
5.2.3	Generating the Ballots	17
5.2.4	Establishing Eligibility and Authentication Details	17
5.2.5	Voting and Auditing Procedures	19
5.2.6	Tallying and Publishing the Results	23
5.2.7	Validation Process	23
5.3	Challenges Encountered	23
5.4	Performance Analysis	25
5.4.1	Ballot Generation	25
5.4.2	User Processes	26
5.4.3	Tallying the Results	26
6	Security Evaluation	27
6.1	Voting System Requirement Analysis	27
6.1.1	Eligibility	27
6.1.2	Unreusability	27
6.1.3	Untraceability	27
6.1.4	Verifiability	28
6.1.5	Tally Correctness	28
6.1.6	Uncoerceability	28
6.1.7	Auditability	28
6.1.8	Accessibility	29
6.1.9	Fairness	29
6.1.10	Soundness	29
6.1.11	Integrity	29
6.1.12	Completeness	29
6.2	Threat Landscape Evaluation	30
6.2.1	A Computationally Limitless Organisation	30
	Threat	30
	Evaluation	30
6.2.2	The Government	31
	Threat	31
	Evaluation	31
6.2.3	Political Parties	31
	Threat	31
	Evaluation	31
6.2.4	Individual Citizens	31
	Threat	31

Evaluation	31
6.2.5 The Anarchist	31
Threat	31
Evaluation	32
7 Conclusion & Further Research Areas	33
7.1 Further Research Areas	33
7.1.1 Proving Ballot Validity	33
7.1.2 Constructing Better Cryptograms	33
7.1.3 Extension to Multiple Candidates	34
7.1.4 Android Secure Implementation Environment	34
7.2 Conclusion	35
 A generate_ballots.java	 36
 References	 41

List of Figures

2.1	Voting Systems	3
4.1	M-Vote Architecture	13
4.2	Authentication Process	14
4.3	Voting Process	15
4.4	Validation Process	15
5.1	Ballot Generation	17
5.2	Authentication Process Diagram	18
5.3	Confirmation String Dialog	20
5.4	Entering the Confirmation String	20
5.5	Selecting a Ballot Option	21
5.6	Auditing a Ballot	21
5.7	Viewing the Receipt	22
5.8	An Example of Published Results	23
5.9	Validating a Vote	24
5.10	Graphs of the Average Ballot Generation and Self-tallying Times	26

List of Tables

3.1	Example Ballot Output	9
5.1	User Process Times	26
7.1	Example Cryptograms	34

Chapter 1

Introduction

1.1 Motivations

In New Zealand, the voter turnout at General Elections is decreasing every election. For instance, in 2005 only 81% percent of enrolled voters chose to cast a vote, then in 2008 this was down to 78%[\[1\]](#). In 2011, New Zealand experienced the lowest turnout under the current system, with approximately 74% of enrolled voters casting their vote[\[2\]](#).

This downwards trend is worrying, as democracy depends on citizens having their say. After the 2011 election, a survey was conducted of 1,097 people, 272 of whom were non-voters. Approximately 23% of those non-voters said that the main reason they did not vote was due to either work or some other commitment, and 14% said they simply could not be bothered[\[3\]](#). This shows that a big contributing factor to the decreasing voter turnout is that people are not wanting, or not able, to make it to a polling station. Because of this, I propose that we should try to take the polling station to the voter.

As of 2013, smartphone penetration in New Zealand was 54%, up from 44% in 2012[\[4\]](#). Most people tend to take their smartphones with them wherever they go. As for those who do not, they would likely ensure that they do have their smartphone on them on election day, if there was a slight chance that they may not be able to make it to a polling station. It is for this reason that I am confident that building a mobile voting application has great potential to increase voter turnout.

It is important to note that I am proposing a mobile application as an additional method of voting, not as a replacement for traditional polling stations. By providing a new way to vote, it is likely that voter turnout will either increase or remain the same, but there is no reason why it should cause a decrease in voters, due to all the traditionally methods

of voting still being available. Therefore, the worst case scenario caused by introducing this M-Vote system is that voter turnout will remain unchanged.

1.2 Project Aims and Scope

The aim of this project is to develop a prototype to test the feasibility of using a mobile voting application for general elections. I will compile a set of voting system requirements which the prototype must meet in order to be considered complete, and also define and analyse a threat landscape in order to evaluate the security of the system. The prototype will be built by extending the principles of remote electronic voting (e-voting), that is, voting remotely via the Internet, to mobile devices.

For this project, the scope will only include the application and the functions of the server. The security and reliability of the mobile phone's hardware and operating system, although important, will not be considered. Due to time limitations, I will only be considering a single-candidate (yes/no style) election and my prototype scope has also been restricted to a scaled-down proof-of-concept.

1.3 My Contributions

My contribution to the field of remote e-voting systems is that I have established that mobile voting, while difficult to accomplish, could be feasible in the future given adequate research and resources. I have proposed various ways to achieve this, and analysed which areas need improvement before such a system could realistically be implemented for a general election.

Chapter 2

Background

There are many different types of voting systems. Most of these systems can be categorised as either paper-based or electronic-based. There are also two main ways in which votes can be cast: In a controlled environment (i.e. on-premises voting), or in an uncontrolled environment (i.e. remote voting). Finally, there are also two main methods of tallying the votes. The first, is that the votes are tallied by a Tallying Authority, whom the public must trust to tally the votes correctly. Alternatively, some systems utilise self-tallying approaches. Figure 2.1 shows the relationship between these different aspects of voting systems, which is discussed in depth in this section.

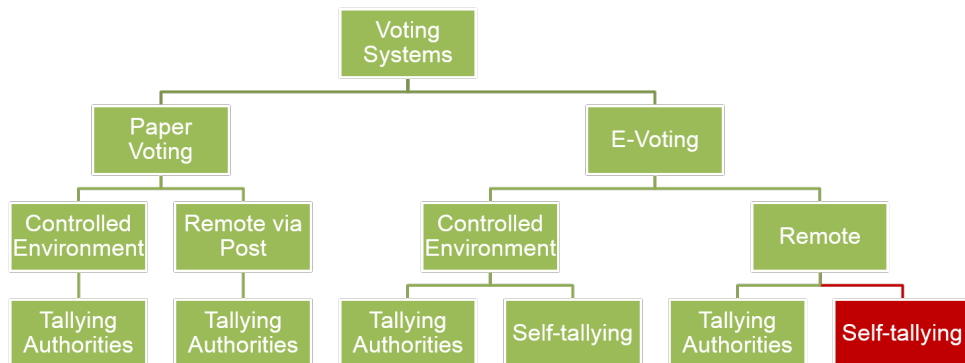


FIGURE 2.1: Voting Systems

2.1 Paper Voting

2.1.1 On-premises

In traditional paper voting systems, a voter must travel to a polling station on Election Day. After proving their identity, they are provided with a paper ballot which they take into a private booth. They record on this paper ballot how they want to cast their vote. They then take this ballot, and place it in a secure, public ballot box. Finally, once the election has ended, the ballots are tallied by the tallying authorities (who are civilians hired for this purpose).

2.1.2 Remote

It is common practice in paper voting systems to allow overseas voters to cast their vote by post. In these situations, the ballot will be posted to the voter a set amount of time before the election, allowing enough time for the ballot to reach the voter, be filled in and then sent back. These ballots are then tallied by electoral officials and added to the totals.

2.2 On-premises E-Voting

2.2.1 Trusted Central Authority

The current state-of-the-art on-premises e-voting process is very similar to paper voting. The key difference being the way in which voters begin the voting process. First, the voter would generally be given an authorisation token, which they take to the private voting booth. This token is presented to the e-voting machine, which then conducts the voting process via a touch screen interface. These systems are reliant on the voters placing trust in the e-voting machine, transitively placing their complete trust in a central tallying authority. On the one hand, involving a tallying authority is nothing new to voting schemes. However, unlike with paper voting, the tallying process cannot be verified by the public. This is due to the fact that everything is done behind the scenes on computers.

2.2.2 Self-tallying

There has been some research[5–8] into the use of self-tallying e-voting frameworks, such that there is no tallying authority involved. Such a framework would, done correctly,

could inspire more trust in e-voting systems, as the voters may have faith that the election cannot be manipulation by malicious authorities.

2.3 Remote E-Voting

The use of personal computers to vote using the Internet (through a web browser) has been trialled in binding local and regional elections in a number of countries around the world[9–14]. These countries include (but are not limited to) Switzerland, France, the Netherlands, Estonia, the United Kingdom and the United States[12].

The country that appears to have made the most progress in regards to Internet voting is Estonia. Since the first successful election involving Internet voting in 2005, the option to vote from home using the Internet has been available in almost all elections, including local, regional and even national elections. The key point that makes this work for them, is that when a voter registers, they are issued a smart identity (ID) card[12, 15, 16] which contains a legal digital signature[17], such that when an election is held, they can vote from any computer that has a smart card reader[18], inserting their ID card as means of authentication[12].

Switzerland has conducted trials over a number of years in elections held at various state levels. Although they have yet to implement Internet voting at a national level, the results from their trials have been largely successful[13].

On the other hand, the United Kingdom has not had as much success. They conducted trials of Internet voting in 2007; however, they deemed them to be unsuccessful due to the cost of the system, being difficult to use for the voter, and a largely unchanged voter turnout. For these reasons, in 2008 they suspended all trials, leaving the future of Internet voting in the UK unclear[11].

In 2010 the United States government developed a pilot project intended to allow overseas absentee voters to use a website portal to vote. In order to test the system before using it in an actual election, they launched the website for a public trial in a mock election. A group from the University of Michigan, posing as attackers, managed to hack in to the system and successfully alter and reveal almost every vote cast within just 48 hours of the trial launch[14]. This trial highlights the importance of security in any remote voting system.

2.4 Related Work

2.4.1 Self-enforcing Electronic Voting

There has been research conducted in the United Kingdom, where they have proposed a Self-enforcing Electronic Voting system[7], such that there is no tallying authority involved in the voting process. In particular, they proposed the DRE-i self-tallying protocol for the Direct-Recording Electronic (DRE) voting systems[7]. This concept forms the basis of this project, and as such will be discussed in depth in chapter 3.

2.4.2 Voting System Requirements

There has been a lot of research over time that has focussed on electronic voting system properties[19]. The requirements proposed in a variety of different papers, of which there were four papers[20–23] that played a pivotal role, have been collated and compiled. What follows is a coherent list of requirements that the proposed mobile voting system should meet:

- REQ 1. *Eligibility*:** Only eligible voters are able to cast a vote.
- REQ 2. *Unreusability*:** Each voter may cast exactly valid one ballot per election. Every ballot is distinct from the others and cannot be used more than once.
- REQ 3. *Untraceability*:** No ballot can be traced back to a specific voter by anyone else, including any authority.
- REQ 4. *Verifiability*:** Voters must be confident that their ballot has been cast as intended.
- REQ 5. *Tally Correctness*:** All ballots are counted correctly. The total amount of all published valid votes must not be more than the number of all registered voters. The voting process is self-tallying, such that no trusted authority is in charge of the tallying process.
- REQ 6. *Uncoerceability*:** A voter cannot prove to anyone else how they cast their ballot. A briber/coercer cannot link any specific ballot to its owner.
- REQ 7. *Auditability*:** A voter may cast an invalid vote for the purposes of auditing.
- REQ 8. *Accessibility*:** Voters are not restricted by physical location from which they can cast their votes. Voters should not require special devices to enable them to vote. Voters should not be required to learn any new skills. Voters should be able to cast their ballots quickly without hassle.

REQ 9. *Fairness*: No one can know the intermediate results of the voting.

REQ 10. *Soundness*: Malicious persons cannot figure out the intention of an eligible voter by intercepting any data transmitted between the voter and the voting authority. No voter can disturb or interrupt the voting process, or affect the result of the voting.

REQ 11. *Integrity*: The voting authority cannot control the election result by some malicious actions.

REQ 12. *Completeness*: All eligible voters are accepted to cast their vote and all ballots are counted correctly.

Chapter 3

The Concept of Self-tallying

3.1 Terminology

3.1.1 Pseudo-Random Numbers

Truly random number are near impossible to randomly generate. This is because in order to generate any number, you require a starting point (called a "seed") which is generally not random. There do exist true random sources, however these are not common and often too expensive to implement. Hence, in this case a *pseudo-random number* generator is utilised, which generates cryptographically secure random numbers.[\[24\]](#)

3.1.2 Public Keys

In cryptography, a *Public Key*[\[17\]](#) is a cryptographic key that is not kept secret and can be used to encrypt messages intended for a particular recipient. In Section [3.2](#), the term refers to the fact that the keys are used to generate the cryptograms for each ballot, which effectively encrypt the ballot option. The terms *Random Public Key* and *Restructured Public Key* are utilised to describe two different keys generated for each ballot. The first term simply refers to the fact that the key is randomly generated, whilst the second term relates to the fact that it has been generated based on all of the random keys.

3.1.3 Finite Cyclic Groups & DDH Intractability

A key concept in Section [3.2](#) is that of *Finite Cyclic Groups*[\[25\]](#) and *Decision Diffie-Hellman (DDH) Intractability*[\[26\]](#). A Finite Cyclic Group is a groupREF (usually

labelled G) that has been generated by a single element, called the *Generator* (usually labelled g). The elements within the group are powers of the generator (i.e. $\{g^0, g^1, g^2, \dots, g^{n-2}, g^{n-1}\}$). The order of the group refers to the number of elements contained within the group (i.e. in the previous example, there were n elements, hence the order was n). If the order of the group is a prime number, it is said to be of *Prime Order*. This last concept is very important when it comes to cryptography. If a Finite Cyclic Group of Prime Order is chosen with the generator g and prime order q carefully chosen to be cryptographically secure and large enough, then DDH Intractability holds.[26] Basically, this means that in order to reverse engineer the keys generated from this group, computational power greater than exists today would be required.

3.2 Ballot Generation

The ballots are generated using a finite cyclic group G with generator g and prime order q . Note the parameters (G, g) and q should be publicly agreed upon before the election.

For each ballot, generate a pseudo-random number, $x_i \in [1, q - 1]$.

Once all x_i have been generated, another number must be generated for each ballot, based on the summation of all the random numbers previously generated using the formula:

$$y_i = \sum_{j < i} x_j - \sum_{i < j} x_j$$

Using these values, we can compute the random public key g^{x_i} , the restructured public key g^{y_i} and the cryptograms:

$$C_i = g^{x_i y_i} g^{v_i} \text{ where } v_i = 0 \text{ for "no", } 1 \text{ for "yes"}$$

Table 3.1 shows the format of the final ballot table. The first three columns are published before the election, but the two cryptograms (last two columns) are kept secret.

Ballot Number	Random Public Key	Restructured Public Key	Yes-cryptogram	No-cryptogram
1	g^{x_1}	g^{y_1}	$g^{x_1 y_1} g$	$g^{x_1 y_1}$
...				
i	g^{x_i}	g^{y_i}	$g^{x_i y_i} g$	$g^{x_i y_i}$

TABLE 3.1: Example Ballot Output

3.3 Cancellation Formula

An important formula that arises from the ballot generation phase is the following cancellation formula:

$$\begin{aligned}
 \sum_i x_i y_i &= \sum_i x_i \left(\sum_{j < i} x_j - \sum_{i < j} x_j \right) \\
 &= \sum_i \sum_{j < i} x_i x_j - \sum_i \sum_{i < j} x_i x_j \\
 &= \sum_{j < i} \sum x_i x_j - \sum_{i < j} \sum x_i x_j \\
 &= \sum_{j < i} \sum x_i x_j - \sum_{j < i} \sum x_j x_i \\
 &= 0
 \end{aligned}$$

3.4 Self-tallying Formula

When a voter casts either a “yes” vote or a “no” vote, the associated cryptogram is recorded. Any ballot that is unused or has only been used for auditing purposes shall be called “dummy” ballots. The self-tallying formula takes all of the cryptograms from the cast votes, along with all of the no-cryptograms from the dummy ballots, and multiplies them together. The output from this product, thanks to the cancellation formula, will be $g^{\#yes-votes}$ as seen by the following cancellation process.

$$\begin{aligned}
 \prod_i V_i &= \prod_i g^{x_i y_i} g^{v_i} \\
 &= \prod_i g^{v_i} \\
 &= g^{\sum_i v_i}
 \end{aligned}$$

When a voter casts either “yes” or “no”, the associated cryptogram is recorded. Once the election is over, the product of all the cryptograms that have been cast (along with all the “no” cryptograms that have not been used) is computed, the output of which will be $g^{\#yes-votes}$.

3.5 Properties of a Self-enforcing Electronic Voting System

In order for a single-candidate electronic voting system to be considered self-enforcing, it should satisfy the following properties:

- SEP 1.** *Well-formedness* - It is easy to verify, given only one of the two cryptograms, that it is an encryption of either "no" or "yes".
- SEP 2.** *Concealing* - Given only one of the two cryptograms, it is infeasible to determine which value ("no" or "yes") the cryptogram represents.
- SEP 3.** *Revealing* - Given both cryptograms, it is easy to determine which cryptogram represents "no" and which represents "yes".
- SEP 4.** *Self-tallying* - Given one arbitrary cryptogram from every one of the N ballots, it is easy for anyone to compute the "yes" tally.

Chapter 4

Proposed Architecture

4.1 The Role of Human Authorities

Traditionally, voting systems involve three major authorities: the Registration Authority (RA), Voting Authority (VA) and Tallying Authority (TA)[5, 27, 28].

I propose that the registration process occurs pre-election and is conducted by the RA as normal. The VA shall oversee the implementation and operation of the servers, but cannot interfere with the voting process itself. It is the role of the TA that is the most complicated part of the M-Vote system.

In most paper-based systems, people are not required to place much trust in the TA, as the ballots are tallied in public. It is this transparency that generates trust in the system. With the shift to e-voting, voters are required to trust the e-voting machines completely. Hence, they must transitively place trust in the TA. As all of this is done behind the scenes, the public have no way to verify that the votes are counted correctly[7].

Even if there is such a thing as a completely trustworthy TA, it is likely that those who make up the TA will not have sufficient technical knowledge to be able to perform the computations involved in initiating the tallying process. This lack of technological knowledge would cause the TA to rely on other, potentially non-trustworthy, persons to initiate the tallying process. In this case, our trust assumption collapses. One such example arose during a trial of the Helios e-voting system[29] when they struggled to find TA trustees who sufficiently fit the criteria.

It is for this reason that I have chosen to make M-Vote self-enforcing, by extending the DRE-i protocol[7] to a remote voting context on a mobile device.

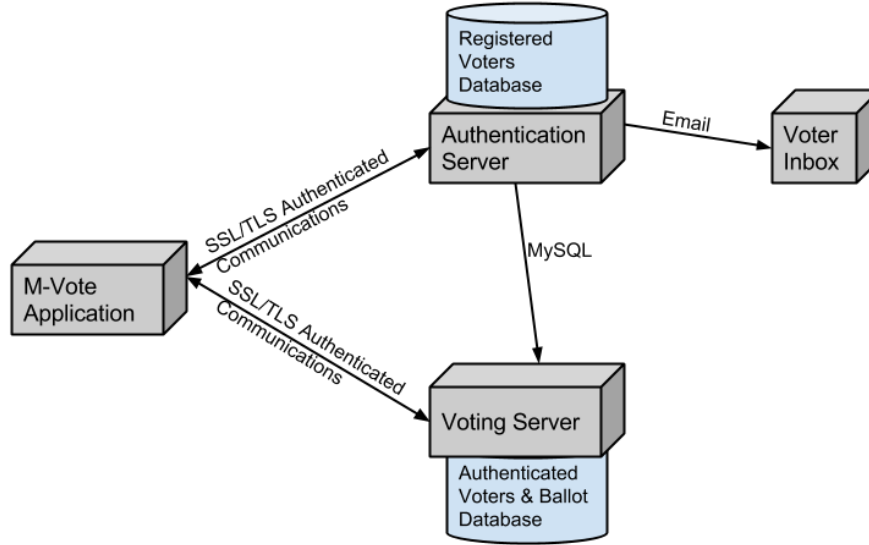


FIGURE 4.1: M-Vote Architecture

4.2 Architecture

4.2.1 Architecture Overview

There are three main components that make up the M-Vote system. First, the M-Vote application itself. Then there are two servers: the Authentication Server and the Voting Server, as shown in Figure 4.1. The Authentication Server has a database containing all the registered voters (including at least a unique Voter ID and an email address). The Voting Server has a database containing all ballots and votes, plus a database containing the Voter ID and Public Key of all authenticated voters. Note that the Authentication Server can access the authenticated voters database on the Voting Server, but the Voting Server cannot access anything on the Authentication Server. All communications between these three components is secured with the Secure Socket Layer (SSL) protocol[30]. The Authentication Server also sends emails to the voter inboxes.

4.2.2 Authentication Procedures

Once the voter has registered offline, they will be assigned a Voter ID and added to the registered voter database by the Registration Authorities. When the voter opens the app for the first time, after having downloaded and installed it via whichever public application store it has been published on, they provide their Voter ID. The Authentication Server sends a randomly generated confirmation string to the associated email address. The voter then enters this confirmation string into the app. Once the Authentication

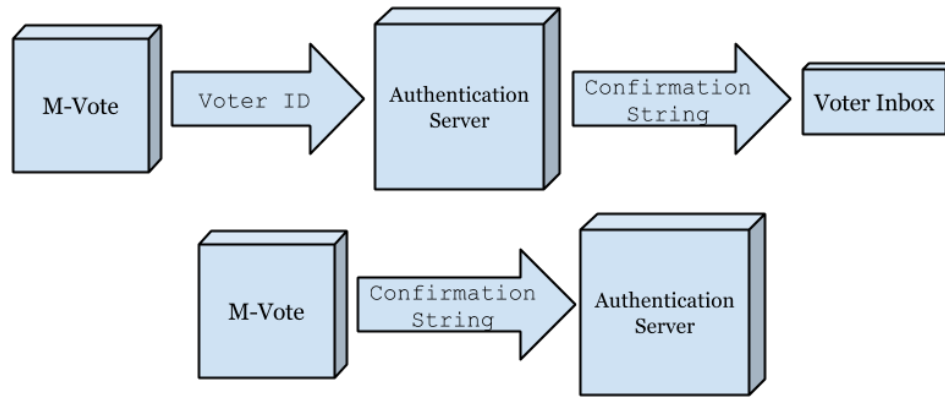


FIGURE 4.2: Authentication Process

Server receives the confirmation string, the voter eligibility has been confirmed. This process is shown in Figure 4.2.

Upon confirmation of eligibility, the M-Vote application will swap public keys with the Authentication Server, such that the ballots may be encrypted on Election Day.

4.2.3 Voting Procedures

On Election Day, the voter opens the app and will be shown a ballot with two options, either “Yes” or “No”. When they press one option, they will be shown the associated cryptogram. They may then either press “Confirm” or “Cancel”. Cancelling the selection will reveal the other cryptogram, effectively auditing the ballot. They will then receive new ballot, and follow the same process. The voter may audit as many ballots as they wish, until they are satisfied that the ballots are legitimate. Once they have confirmed a selection, the ballot is sent to the server and they will be shown a receipt which lists all the audited ballots and the confirmed ballot with the selected cryptogram. Figure 4.3 shows the voting process.

After the election has ended and the results have been published, the voter may validate that their vote has been counted. This may be done one of two ways. The easiest, is via the M-Vote application. When they open the application, they will be given the option to validate their vote. This will automatically look up their cryptogram to ensure it is on the published list of votes. They also have the option to manually verify this, if they do not trust the application to do it for them. Given the cryptogram from their receipt, they can search through the published votes to ensure it is there. This would be a very tedious process in a general election, due to the huge number of votes, but it must be an option in order to be transparent. Figure 4.4 shows the validation process.

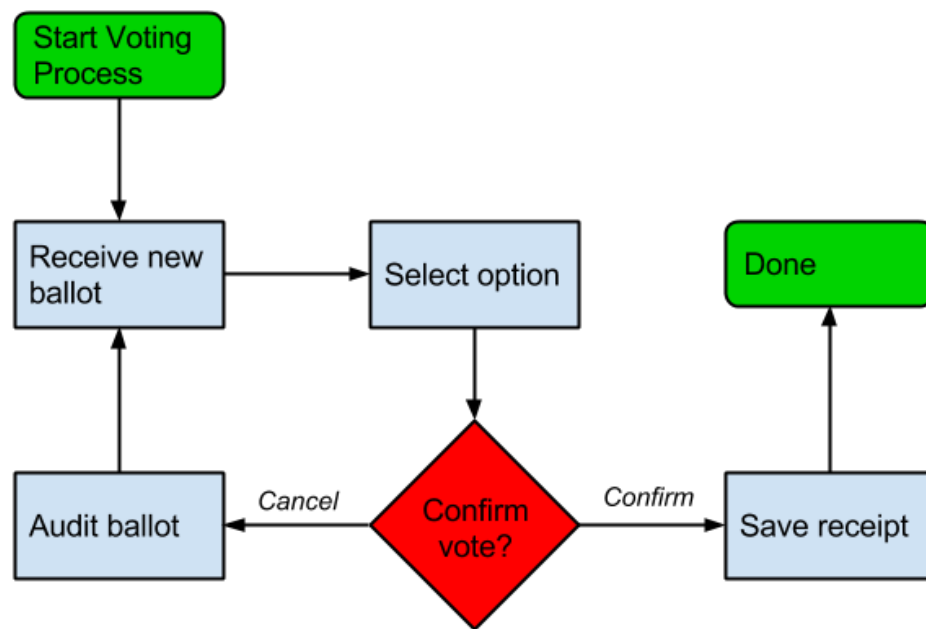


FIGURE 4.3: Voting Process

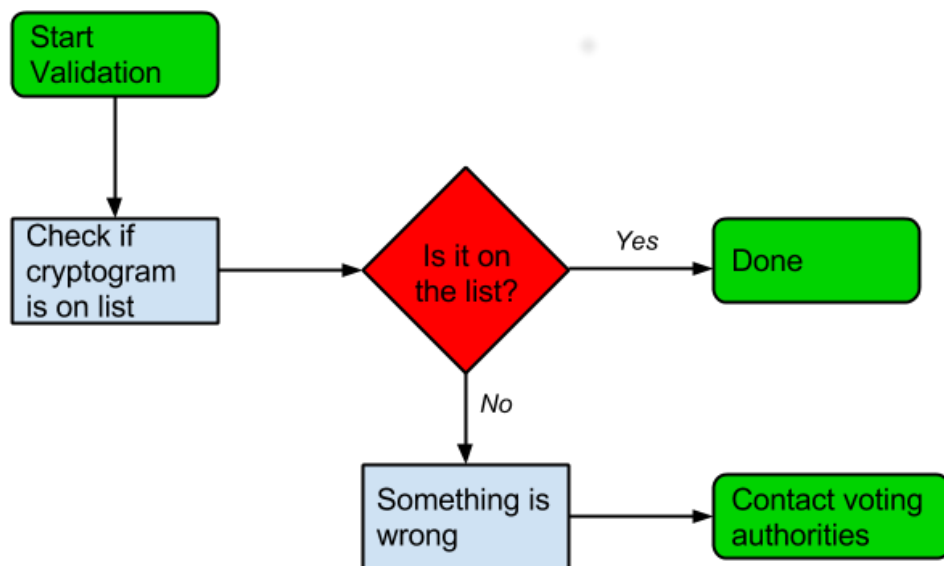


FIGURE 4.4: Validation Process

4.3 Challenges: Coercion

In any form of remote voting, the votes are cast from an unsecure location, where a coercer could be standing over the voter's shoulder to ensure that the vote is cast as they intend it to be cast[31, 32]. For this reason, it is likely that the problem of coercion might be extremely difficult to resolve completely in the context of remote electronic voting.

Chapter 5

Implementation

5.1 Proof of Concept Overview

Due to time constraints, it was not feasible to develop a prototype for a General Election context. For this reason I decided to develop a proof-of-concept prototype, modelling a University Election where all enrolled students are eligible to vote. This prototype works for either a Yes/No style election, or a two-candidate election. The proof-of-concept assumes that there is an existing database of all students, containing at least the Student ID and email address of all enrolled students.

The success of the prototype will be determined based on an analysis of how well it satisfies the Voting System Requirements (Section 2.4.2), as well as an analysis of the system according to the Threat Landscape (Section 6.2). I claim that the success of this prototype proves the feasibility of a full-scale General Election prototype, as when scaling the prototype. The reason for this is that all requirements should still be met, provided the necessary steps for improvement, suggested in the analysis in chapter 6, are taken.

5.2 Implementation

5.2.1 M-Vote Application

Android 4.0[33] was the chosen platform for this prototype, using SpongyCastle 1.50[34] as the encryption library. SpongyCastle is a variation of BouncyCastle[35], which has been optimised for Android development.

```
brandon@m-vote-voteserv:~$ java -cp ./lib/* generate_ballots 1000
Generating 1000 ballots...
[YES] Cancellation formula holds.
Ballots generated.
```

FIGURE 5.1: Ballot Generation

5.2.2 Creating the Servers

M-Vote consists of two servers: an Authentication Server (AS) and a Voting Server (VS). Both of these servers are running Ubuntu Server 14.04[36, 37] with MySQL 5.5.37[38] and are written using Java 1.7[39, 40]. For consistency, SpongyCastle has also been utilised on the servers. In order to conduct email communication, the AS uses the JavaMail library (version 1.4.5)[41].

5.2.3 Generating the Ballots

The ballots are generated before the election begins. This is done on the VS and should be initiated by the Voting Authorities as seen in Figure 5.1. For this prototype, the standard 2048 bit DSA group settings[42] were used for the finite cyclic group used to generate the ballots. With these settings, DDH intractability holds, as required in Section 3.2.

The generate_ballot Java file (Appendix A) simply implements the algorithm discussed in Section 3.2. The number of ballots generated should be significantly larger than the voting population, in order to account for auditing[6].

5.2.4 Establishing Eligibility and Authentication Details

When the application is first opened, an SSL connection is established between the app and the AS, using SSLSockets[43]. This secure channel is used throughout the entire eligibility and authentication establishment process, of which a diagram is shown in Figure 5.2.

- 1) The user should already be a registered voter (this is done offline and outside of the scope of the system).
- 2) The application is downloaded and installed from the appropriate app store.
- 3) When the application is opened, the app requests the user's student ID.
- 4) The user enters their student ID into the application.

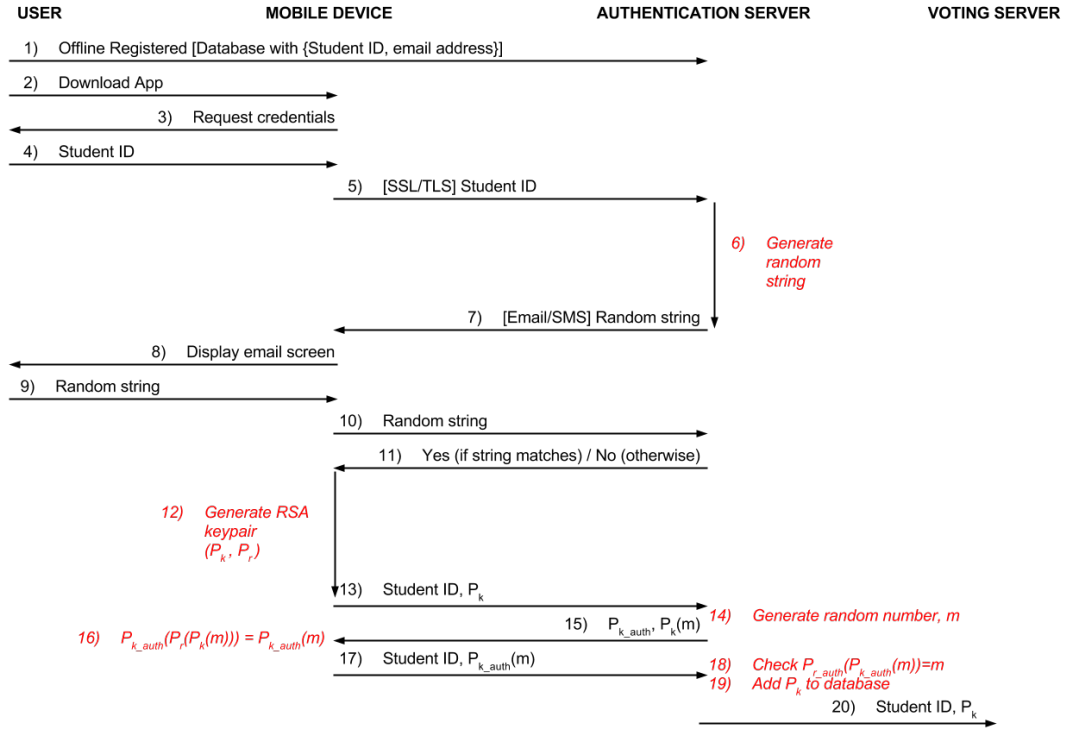


FIGURE 5.2: Authentication Process Diagram

- 5) The application sends the student ID to the AS (using the previously established SSL connection), requesting that the AS check whether the student ID belongs to a valid voter.
- 6) If the ID is valid, then the AS generates some random bytes (called the "confirmation string"). If the ID is invalid, a response is sent to the application saying so.
- 7) The confirmation string is sent to the email address that is associated with the student ID (this is found in the database entry created during the registration process). A response is sent to the application, informing it that the confirmation string has been generated and sent.
- 8) If the application receives a response saying the ID is invalid, the user is informed and the process stops. Otherwise, the app will ask the user whether they would like to open the email application on their device. This process is shown in figure 5.3.
- 9) Either by copying from the email message in the email app, or by manually entering the string after finding it on another device, the user will provide the app with the confirmation string as shown in figure 5.4.
- 10) This confirmation string is sent to the AS by the app, requesting that the string be confirmed.

- 11) If the confirmation string is correct, then the user's eligibility has been established. Otherwise, the user is likely ineligible. A response is sent to the app informing it of the outcome.
- 12) Now that the eligibility has been established, the final stage is to share credentials. The app will now generate an RSA keypair.
- 13) The app sends the public key (along with the student ID) to the AS.
- 14) The AS generates a random number.
- 15) The random number is encrypted using the public key from the app, and sent to the app along with the server's public key.
- 16) When this response is received, the app will decrypt the random number using its private key, then re-encrypt the number using the server's public key.
- 17) The re-encrypted number is sent back to the AS (along with the student ID).
- 18) The AS decrypts the random number using the server private key, then checks that the number is the same random number generated earlier.
- 19) Now that the credentials have proven to be valid, the app public key is stored in the database with the student ID.
- 20) Finally, the AS also adds this public key and student ID combination to the database on the VS, such that it may be used during the voting process. Whenever the app is opened in the future, it will automatically identify the voter to the AS and the voter will never have to go through the authentication process again.

5.2.5 Voting and Auditing Procedures

During the election, when the app is opened it sends a request to the VS asking for a ballot. The VS retrieves the first unused ballot from the database, flags it as having been used, and sends it to the app. The voter may now choose one of the two options by pressing the button, as shown in Figure 5.5.

The cryptogram from the selected option is now displayed to the voter, on the button itself. Now the voter has the option to either confirm the vote, or cancel it. Should they choose to cancel, the remaining cryptogram is displayed in the same manner, as can be seen in Figure 5.6. The voter has now effectively performed an audit on the ballot, as they can see the cryptograms of both options, and it is easy to see that the "yes"

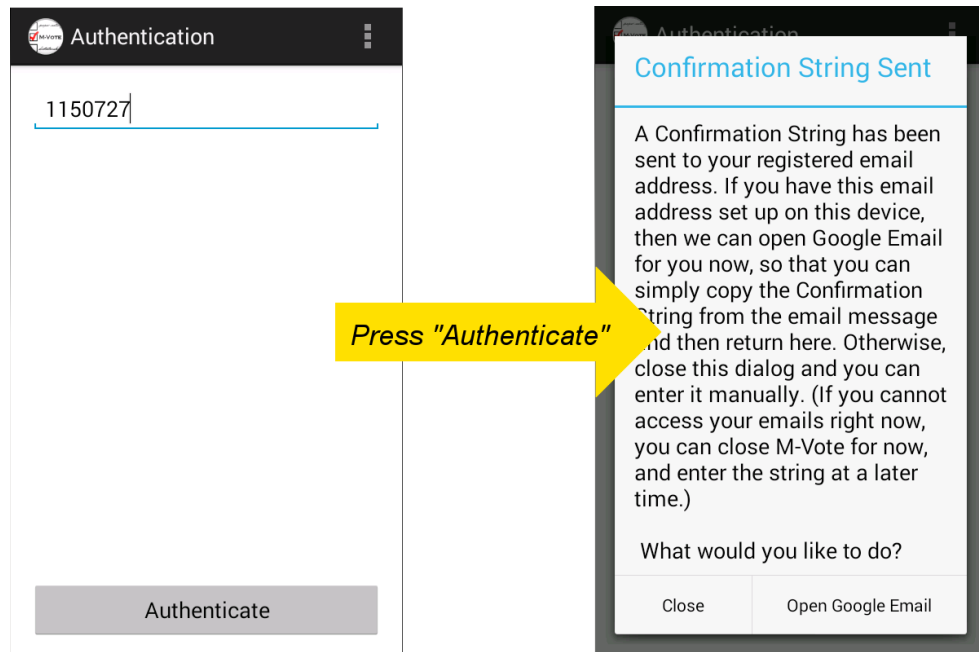


FIGURE 5.3: Confirmation String Dialog

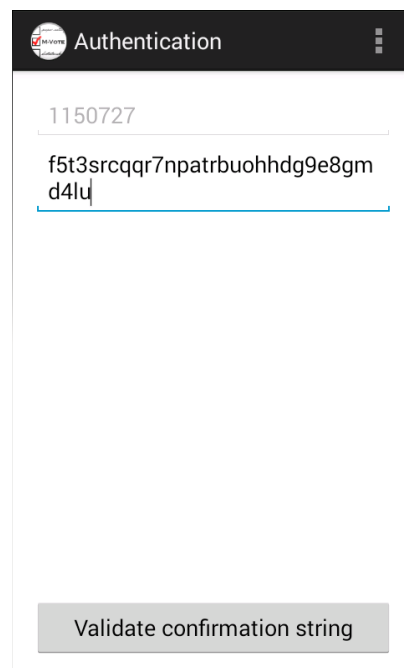


FIGURE 5.4: Entering the Confirmation String

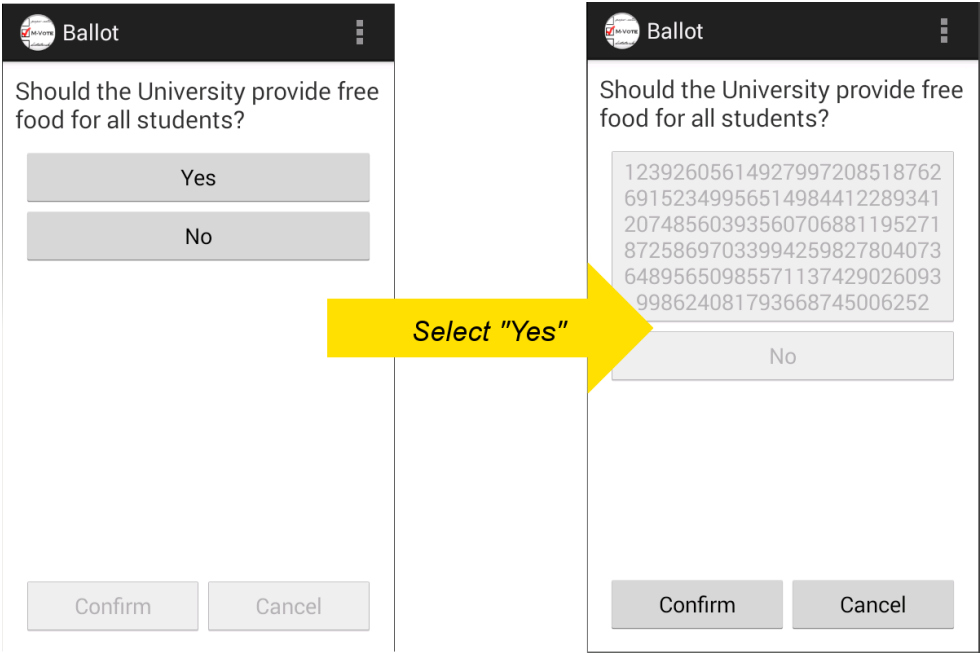


FIGURE 5.5: Selecting a Ballot Option



FIGURE 5.6: Auditing a Ballot

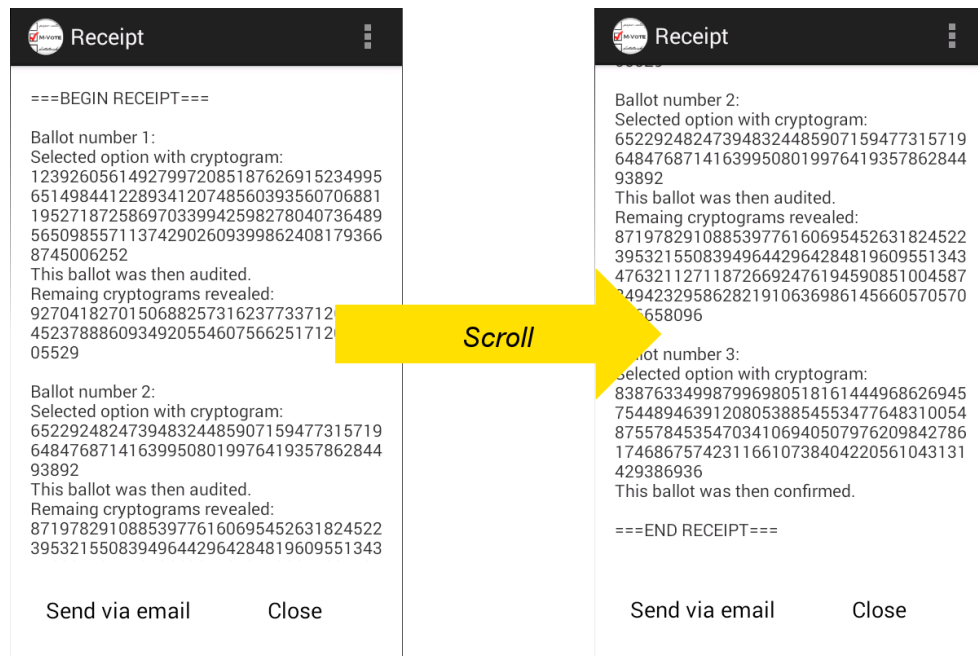


FIGURE 5.7: Viewing the Receipt

cryptogram is much bigger than the “no” cryptogram, as would be expected due to the way in which the cryptograms have been constructed (see Section 5.2.3).

Once the voter is satisfied that the ballot is legitimate, they can load a new ballot. When this happens, the app sends the ballot number and a message saying it has been audited to the VS, who records the fact that it was an audited ballot, and then sends a new ballot to the app as discussed previously. This auditing process may be repeated as many times as they wish, until they are satisfied that the system is behaving as it should be. Note that in practice, a limit should be enforced, such that the ballots do not run out.

Eventually, the user will select their chosen option and confirm the vote. When this is done, the app sends the ballot number, the cryptogram of the selected option and the student ID to the VS. The cryptogram is recorded along with all the other votes, the ballot number is recorded as having been confirmed, and the student ID is flagged as having voted. Finally, a “finished” message is sent back to the app, who displays the receipt to the voter as seen in Figure 5.7. The receipt is also stored within the app, so it can be accessed at a later date, and there is an option to send the receipt to the voter’s email address (via the AS).

```

3,valid,83876334998799698051816144496862694575448946391208053885455347764831005487557845354703410694050797620
9842786174686757423116610738404220561043131429386936

1,dummy,92704182701506882573162377337126802045237888609349205546075662517120236905529,12392605614927997208518
7626915234995651498441228934120748560393560706881195271872586970339942598278040736489565098557113742902609399
8624081793668745006252

2,dummy,65229248247394832448590715947731571964847687141639950801997641935786284493892,87197829108853977616069
5452631824522395321550839496442964284819609551343476321127118726692476194590851004587849423295862821910636986
145660570570826658096

...

1000,dummy,65997133699560584861008579586688355876627539161616829661420401561313844764152,88224330965493089713
5247570722094946870516197485455980065410727229060976140723928655563511232161742822416369317182113062324496391
715674206579251833858976

===ENDOFCRYPTOGRAMS===

Self-tallying result string: 13367903425490809705830959442499616299991505654103128955029129184013231509388
Verification string: 13367903425490809705830959442499616299991505654103128955029129184013231509388

RESULTS: Yes (1), No (0)

Final results verified.

```

FIGURE 5.8: An Example of Published Results

5.2.6 Tallying and Publishing the Results

At the end of the election, the votes are tallied and the results published. The VS uses an implementation of the tallying algorithm specified in Section 3.4. For this prototype, the results are only published to a file, however in a real-world context they should be published somewhere like a bulletin board. Figure 5.8 shows an example of the published output.

5.2.7 Validation Process

Once the results have been published, it is simple for any voter to validate that their vote has been counted. M-Vote enables this process to be automated (as seen in Figure 5.9). When the voter signals that they wish to validate their vote, the application sends the cryptogram to the VS, which searches through the list of published cryptograms. The VS then returns the success/failure of the search to the application, which in turn tells the user.

If the voter wishes to, they may manually validate their vote by comparing their cryptogram to the published results.

5.3 Challenges Encountered

The main challenges I faced throughout this project, is that before I started I had very little knowledge about cryptography, which made the research process a lot slower than I had hoped, seeing as most papers that discusses e-voting protocols use some form of cryptography, which varies from one paper to the next.

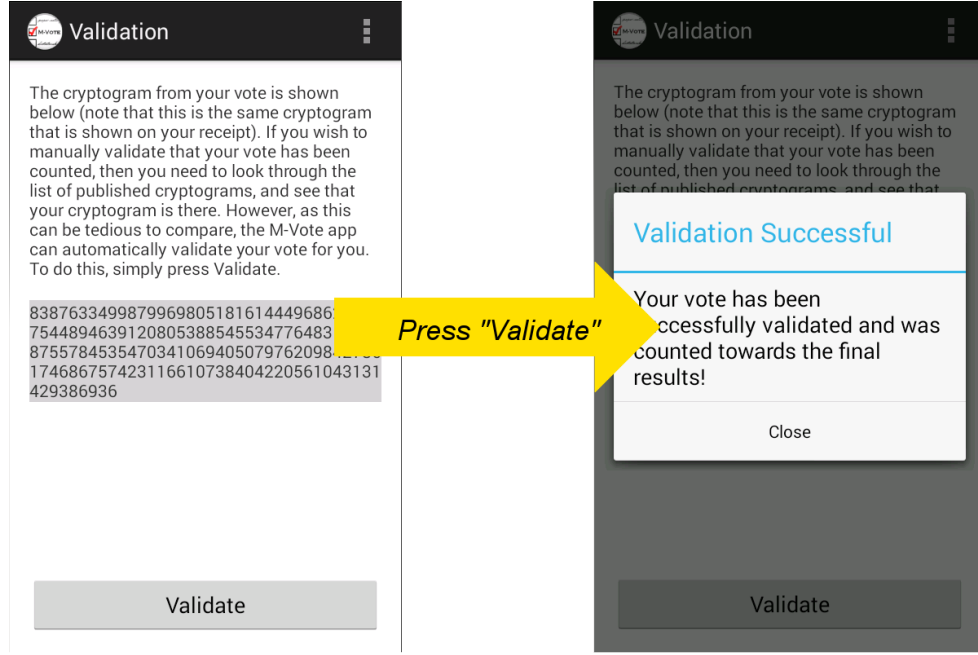


FIGURE 5.9: Validating a Vote

When it came time to develop the prototype, it took me a long time to get going properly, due to the fact that I had never done any Android development prior to this project. Therefore I had to teach myself how to develop Android applications from scratch at the same time as actually building the prototype. I did not foresee this being much of an issue, as I have plenty of programming experience. However it turns out that Android development, although it is mostly done in Java, is done quite differently to the standard Java development that I am used to.

As for the generation of the restructured public keys, the equation in [6] is:

$$g^{v_i} = \prod_{j < i} g^{x_j} / \prod_{i < j} g^{x_j}$$

However, this equation will end up with the first 50% of the keys being of the form $g^{(negative-number)}$. This is not a problem mathematically, as it is calculated using modulo exponentiation. But when implementing this equation in Java, it simply gives 0. Naturally, this is not very useful. As such, the v_i values (and hence the restructured public keys) were calculated as discussed earlier.

There was one issue in particular that stalled the prototype development. As mentioned earlier, all communications were conducted over an SSL channel. In general, establishing such a connection is almost trivial. However, as my servers were using self-signed certificates, it became a much bigger problem, because Android applications in general will only accept certificates that have been issued by a trusted Certificate Authority[44].

Many different solutions have been published on the Internet, however the majority of them simply do not work. It took a long time to find a solution to this problem, which in reality is not actually that hard to solve. All that is required is to import this certificate into a custom TrustManager[45], as can be seen in Listing 5.1.

```
// Load the server keystore
KeyStore keyStore = KeyStore.getInstance("BKS");
keyStore.load(ctx.getResources().openRawResource(R.raw.mvotekeystore), "PASSWORD"
    .toCharArray());

// Create a custom trust manager that accepts the server self-signed certificate
TrustManagerFactory trustManagerFactory = TrustManagerFactory.getInstance(
    KeyManagerFactory.getDefaultAlgorithm());
trustManagerFactory.init(keyStore);

// The same for key manager
KeyManagerFactory keyManagerFactory = KeyManagerFactory.getInstance("X509");
keyManagerFactory.init(keyStore, "PASSWORD".toCharArray());

// Create the SSLContext for the SSLSocket to use
SSLContext sslctx = SSLContext.getInstance("TLS");
sslctx.init(keyManagerFactory.getKeyManagers(), trustManagerFactory.
    getTrustManagers(), new SecureRandom());

// Create SSLSocketFactory
SSLSocketFactory factory = sslctx.getSocketFactory();

// Create socket using SSLSocketFactory
SSLSocket client = (SSLSocket) factory.createSocket(serverAddress, 8081);

// Print system information
System.out.println("Connected to server " + client.getInetAddress() + ": " +
    client.getPort());
```

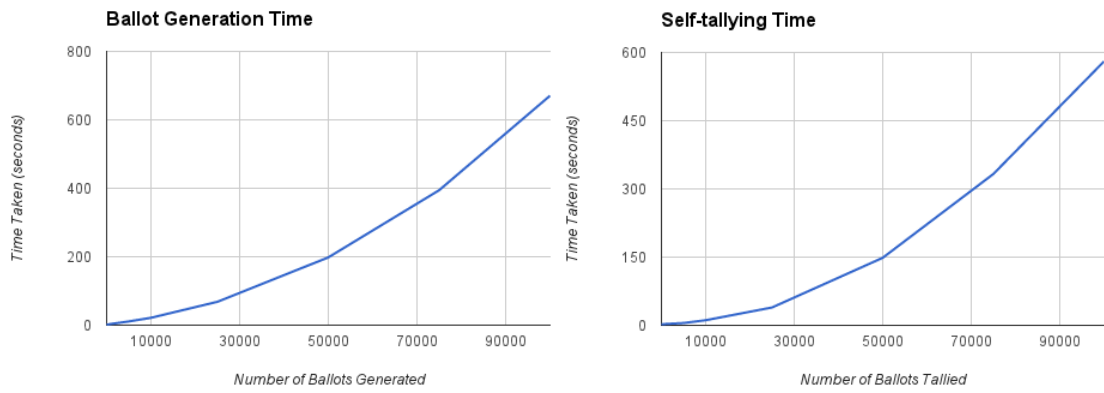
LISTING 5.1: SSL Connection Establishment Code Extract

5.4 Performance Analysis

It is important to note that this performance analysis is only a guideline, and I would expect a full scale prototype to perform at a greater speed, given that I have spent no time on optimising the performance of the proof-of-concept.

5.4.1 Ballot Generation

Figure 5.10a shows a graph that plots the average time taken to generate a set number of ballots, up to 100,000. As can be seen, the trend looks like that of a polynomial.



(A) Average Ballot Generation Time Graph

(B) Average Self-tallying Time Graph

FIGURE 5.10: Graphs of the Average Ballot Generation and Self-tallying Times

5.4.2 User Processes

Next we look at the basic time taken from the user's perspective to complete each of the processes involved. Note that these figures are clearly subjective, and depend on the Internet speed at the given time, as well as the user's technological literacy level. Table 5.1 shows the times for one user.

Process	Time Taken
Eligibility & Authentication	1 minute 10 seconds
Auditing a Ballot	14 seconds
Casting a Vote	9 seconds
Validating a Vote	8 seconds

TABLE 5.1: User Process Times

5.4.3 Tallying the Results

Figure 5.10b shows a graph that plots the average time taken to complete the self-tallying process for an election with a set number of ballots, up to 100,000. Due to the way in which the self-tallying protocol works, an election with n total ballots, will take the same tallying time regardless of the number of ballots actually used. As can be seen, the trend looks like that of a polynomial.

Chapter 6

Security Evaluation

6.1 Voting System Requirement Analysis

The following is an analysis of each voting system requirement.

6.1.1 Eligibility

During the eligibility establishment phase, the voter must provide an eligible student ID and have access to the associated email address. If this is the case, it is almost certain that they are indeed eligible to vote. Of course, it is possible that the voter found an eligible student ID. However, they would also need to find out the email address associated with it, and then maliciously gain access to the inbox. This problem would be much bigger than just a voting issue, and as such has not been addressed.

6.1.2 Unreusability

As soon as a voter has cast their vote, the Voting Server records that their ID has voted. This ID will no longer be allowed to vote, preventing a single person from voting more than once. Due to the way in which the ballots are constructed, the probability of any cryptograms being the same is negligible.

6.1.3 Untraceability

When the Voting Server receives a vote, the cryptogram is added to the vote database and the authentication database is updated to reflect that the ID has been used to cast a

vote. While the Voting Server performs these actions, they could easily determine how a particular ID had voted, but since it cannot access the Authentication Server database, there is no way to determine who the ID belongs to. The Authentication Server can access the Voting Server database, but as no link between the ID and the cryptogram is ever stored, it can only see who has voted, but not how they voted. The prototype for this project has been implemented such that these conditions are satisfied, however in a real world context, there will have to be an assumption of trust, by the voters, that the communication restrictions are enforced correctly.

6.1.4 Verifiability

Due to the auditability requirement (REQ 7), voters can be confident that all votes are being cast as intended. As the votes are published post-election, they may also verify that their individual votes has indeed be counted. These two features combine to provide strong verifiability.

6.1.5 Tally Correctness

The public may have complete faith that the tally is correct, providing that the self-tallying output matches the computed value of $g^{\#yes-votes}$. This is discussed in more depth in Section 3.4.

6.1.6 Uncoerceability

After a voter has cast their vote, there is no way that they may prove to a coercer how they voted. This is due to the DDH intractability[25] (discussed in Sections 3.1, 3.4 and 5.2.3). The votes cannot be traced back to the voter, as there is never any link recorded once the vote has been cast. Unfortunately, this requirement may never be solved completely for a remote voting system. This is because the environment is uncontrollable, unlike polling stations. There is no way to prevent coercers from standing over a voter's shoulder while they vote.

6.1.7 Auditability

The voters may audit as many ballots as they wish. This is done by showing both cryptograms, allowing the voter to verify that the “yes” cryptogram is indeed larger than the “no” cryptogram (as it should be, based on the ballot generation algorithm).

However, there is still a hole in this requirement, in that the voter has no way to know that the cryptograms contain a valid vote. This issue is addressed in Section 7.1.1.

6.1.8 Accessibility

Any mobile device that meets the minimum requirements may be used to cast a vote. (Note that this prototype is only available for Android 4.0+, however the concept can be easily implemented for many versions of all device platforms.) The physical location of the voter does not affect whether or not they can vote, providing that they have either Wi-Fi or Mobile Data access. The voting process itself takes minimal time and effort, as described in Chapter 5.

6.1.9 Fairness

The tallying process cannot be run while the election is in progress, hence the partial results cannot be known without interrupting the voting process, which would be publicly obvious.

6.1.10 Soundness

There is no way that a malicious person could determine how any voter has voted, for two reasons. First, the communications are encrypted. Should the encryption be broken, it is still impossible (with the current computational power) to reverse engineer a cryptogram to determine which vote it contains, as discussed in Chapter 3. If a malicious person managed to find a way to alter the results, which in itself is unlikely, it would not go unnoticed due to the nature of the self-tallying and verifiable protocols used.

6.1.11 Integrity

As the votes are self-tallied, and this result can be easily verified by the public, any malicious actions would be blatantly obvious to any person who were to verify the published results. Hence the system has integrity.

6.1.12 Completeness

This requirement is clearly satisfied due to both the eligibility (REQ 1) and tally correctness (REQ 5) requirements being met.

6.2 Threat Landscape Evaluation

A voting system will always have a very wide threat landscape. I will evaluate a wide variety of these threats, from a range of different entities. However a much more extensive evaluation would be required before this system could be considered sufficiently secure.

6.2.1 A Computationally Limitless Organisation

Threat This organisation could want to manufacture the election results, create suspicion, or even prevent the election from occurring.

Evaluation Any entity that has enough computational power to break all encryption could perform any number of attacks on the election. If their aim is to create suspicion or derail the election, there is nothing that can be done to prevent this from happening to the M-Vote system.

If the aim is to manufacture the election results, whilst ensuring the public believe the results are valid, this would be much more difficult. There are two main ways in which the result could be tampered with. The first would be by not altering any votes, but instead adding many more votes in favour of the chosen candidate. Once the Voting Server has been hacked, this is a simple process. Just take a whole lot of the dummy ballots and add the cryptogram of the chosen candidate to the vote database. If the voter turnout is significantly low, it is likely that this attack could go unnoticed, as long as the number of published valid votes does not exceed the number of enrolled voters. However, if there is a high voter turnout then this attack would either be noticed (by the number of published votes exceeding the number of voters) or have little effect, due to not being able to add enough votes to change the outcome. Of course, if it is a very close election, a few votes may be all that is needed.

The alternative would be to alter the votes as they are cast. Seeing as the computational power is unlimited, it would not be hard to do this. This attack is unlikely to succeed, assuming most people validate their vote, as too many people would have validation fail. Of course, it would not be hard to alter the server to tell people that their vote was validated successfully even if it was not. In this case, manually validating votes (by looking up the cryptogram on the published list) would reveal the fraud, but only if a lot of voters did this, which is unlikely.

6.2.2 The Government

Threat The current Government naturally want to remain in power, and so could try to fix the outcome in their favour.

Evaluation If the Government control the previously mentioned Computationally Limitless Organisation, then that organisation would be their best chance at fixing the election outcome, hence the same analysis applies. However, if they do not, then there is no way in which they could alter the outcome, without it becoming obvious during public verification.

6.2.3 Political Parties

Threat The political parties who are contesting the election, could benefit by knowing the intermediate results of the election.

Evaluation This is not possible, thanks to the fairness requirement (REQ 9) that was proven in Section [6.1.9](#).

6.2.4 Individual Citizens

Threat Individuals who feel strongly about a particular candidate may try to coerce voters either by force or bribery.

Evaluation The M-Vote system has been designed such that a voter cannot prove how they voted after the fact. This means that in order for coercion to be successful, the coercer must be present at the time of the vote, such that they can watch the actual vote being cast. Due to the remote nature of the M-Vote system, as discussed in Section [6.1.6](#), this cannot be prevented. However, in order for a coercer to have an impact on the final result, they would have to successfully coerce a large number of voters, which would be difficult to do within the short timeframe of an election, unless there was a large group of coercers working together.

6.2.5 The Anarchist

Threat The anarchist wants only to nullify the election, by creating doubt around whether the results can be trusted or not.

Evaluation There are two ways in which this goal could be achieved. The first is by taking down the Voting Server, perhaps by some sort of denial-of-service attack. Good server implementation will reduce this risk, however as this is outside the scope of this project, it will not be covered. The second way in which the anarchist may achieve their goal is by claiming that their validation failed. This is unlikely to work, for several reasons. They would need to provide their receipt, so as to prove that the validation failed. Providing a receipt for a vote that actually was counted would be useless, and their argument would be over. Hence they would have to create a fake receipt that “proves” that their vote was not counted. This attack may work, if they were to take a cryptogram from one of the published dummy ballots, saying that their receipt shows that their vote was cast, but on the published results it was counted as a dummy ballot. At this stage, I am not sure of a way to prevent this type of attack from occurring. However, if every other voter successfully validates their vote, the attack would not achieve the desired effect of creating suspicion. Hence it would only work if a large number of people were involved in the plan.

Chapter 7

Conclusion & Further Research Areas

7.1 Further Research Areas

7.1.1 Proving Ballot Validity

Right now, there is no way that the voters can be confident that the individual cryptograms are valid. That is, they cannot be sure that any given cryptogram is actually of the form $g^{x_i y_i} g^{v_i}$ with $v_i = 0$ or 1 . If we can find a way to provide such an assurance, only then will the auditability requirement be completely satisfied. The use of zero-knowledge proofs has been suggested[6] as a solution to this problem. There may be other ways, however it is important that however this assurance is provided, the other requirements (such as uncoerceability) are not broken as a result.

7.1.2 Constructing Better Cryptograms

The downside to using the 2048 bit DSA group setting for the ballot generation (as discussed in Section 5.2.3), is that the two cryptograms end up having significantly different size. For instance, a typical ballot generated by this system is shown in Table 7.1.

If a person were to audit several ballots, it would quickly become apparent that the “no” cryptograms are quite short and the “yes” cryptograms are quite long. It is, of course, important for auditability that the “yes” cryptograms are always larger than the “no” cryptograms, however with such a degree of size difference it would become quite easy

Cryptogram of “no” vote	Cryptogram of “yes” vote
36770592302101863369919632394524	49154572679259349763265545318178
57837085251698352744927088116257	18701871662642689519306867691133
7029185508581	71710907649159074231773229822711
	04475097790051391900361986572385
	74905469110783056058428

TABLE 7.1: Example Cryptograms

to determine whether any given cryptogram represented “yes” or “no”, which breaks the uncoerceability requirement.

Note that a significantly smaller generator would provide cryptograms of much similar length, however we would then lose the DDH intractability and as such, our uncoerceability requirement would still remain broken.

So whilst we have theoretically satisfied the uncoerceability requirement via mathematically sound arguments, in practice this is unlikely to be feasible. If a way is found that we may generate cryptograms of similar size whilst not breaking the auditability requirement, then the system would satisfy practical uncoerceability as well as theoretical.

7.1.3 Extension to Multiple Candidates

In order for this system to work in a General Election context, it needs to be extended to allow multi-candidate elections. There have been several ways suggested[5, 6, 27, 46] which may be feasible, however due to time constraints it was not possible for me to research too deeply.

7.1.4 Android Secure Implementation Environment

The biggest flaw with this system is the Android platform itself. Due to the lack of a secure implementation environment, all Android applications are susceptible to malware. Because of this, it is theoretically possible for a malicious person to create malware that, once spread across many smartphones, could hijack the election by interfering with the voting application. Hence it is important that research into securing the environment is conducted.

7.2 Conclusion

I have created a proof-of-concept that demonstrates that a mobile voting system is indeed feasible for a general election context. There is a lot of research and development required before it can realistically be utilised, and I have pointed out some of these key areas. The most important area that must be well researched and implemented, is the security of the platform and the servers. Without such security, any remote voting system should not be considered.

Aside from the technology issues, one of the main obstacles that must be overcome before such a system is accepted, is that of understandability. How can we get the general public to trust a system that they do not completely understand? This problem is one that faces all e-voting systems, but becomes even more of an issue when the basis of the system is complex cryptography, which the average person would have no knowledge of. Unfortunately, this is quite a difficult problem to overcome, and I do not have the answer.

There are many ways in which one could deal with human authorities in a remote e-voting system, however I strongly believe that some sort of self-tallying system is the right way to go, as the reduction of human involvement will inspire trust in the system. The less control human authorities have over the results, the less chance that they could interfere with the process (willingly or not) and as such there will be less chance for suspicion to be cast over the election.

Finally, I do not believe that any remote voting system should replace traditional polling station voting, as technology is never going to be completely reliable. A mobile voting system should be used in addition to any other existing methods. The more ways there are to vote, the greater chance of increased voter turnout.

Appendix A

generate_ballots.java

```
1 import java.io.BufferedWriter;
2 import java.io.File;
3 import java.io.FileOutputStream;
4 import java.io.OutputStreamWriter;
5 import java.math.BigInteger;
6 import java.security.SecureRandom;
7 import java.sql.Connection;
8 import java.sql.DriverManager;
9 import java.sql.PreparedStatement;
10 import java.sql.ResultSet;
11 import java.sql.SQLException;
12 import java.util.ArrayList;
13 import java.util.List;
14
15 /**
16  * Generates ballots for use in an election using the M-Vote app.
17  * Note that this class will need to be customised for each election.
18  *
19  * @author Brandon Nicholson
20  */
21 public class generate_ballots {
22
23     // Global constants
24     private final static int MINIMUM_BALLOTS = 1;
25     private final static String qAsBase16 = "F2C3119374CE76C9356990B465374A17F23F9E
26         D35089BD969F61C6DDE9998C1F";
27     private final static String gAsBase16 = "5C7FF6B06F8F143FE8288433493E4769C4D988
28         ACE5BE25A0E24809670716C613D7B0CEE6932F8FAA7C44D2CB24523DA53FBE4F6EC3595892D1
29         AA58C4328A06C46A15662E7EAA703A1DECF8BBB2D05DBE2EB956C142A338661D10461C0D1354
30         72085057F3494309FFA73C611F78B32ADBB5740C361C9F35BE90997DB2014E2EF5AA61782F52
31         ABEB8BD6432C4DD097BC5423B285DAFB60DC364E8161F4A2A35ACA3A10B1C4D203CC76A470A3
32         3AFD CBDD92959859ABD8B56E1725252D78EAC66E71BA9AE3F1DD2487199874393CD4D832186
33         800654760E1E34C09E4D155179F9EC0DC4473F996BDCE6EED1CABED8B6F116F7AD9CF505DF0F
34         998E34AB27514B0FFE7";
35
36     // Global secure random number generator
```

```

29     private static SecureRandom random = new SecureRandom();
30
31     // Pre-election key publishing file
32     private static final String PUBLIC_KEYS_FILE_NAME = "./published/
        ballots_and_public_keys.mvp";
33
34     /**
35      * Main: Controls generating the ballots.
36      * @param args Parameters.
37      */
38     public static void main(String[] args) {
39         // Checks that we have sufficient arguments
40         if (args.length < 1) {
41             display_usage();
42         }
43         else {
44             try {
45                 // Get the number of ballots
46                 int num = Integer.parseInt(args[0]);
47                 if (num < MINIMUM_BALLOTS) {
48                     System.err.println("The number of ballots to generate must be an
        integer of at least "+MINIMUM_BALLOTS+".");
49                 }
50
51                 // Generate ballots
52                 generate(num);
53
54             } catch (NumberFormatException e) {
55                 // Catch any attempt to access a "--help" style flag
56                 if (args[0].equals("-h") || args[0].equals("-help") || args[0].equals("--
        h") || args[0].equals("--help") || args[0].equals("/?")) {
57                     display_usage();
58                     return;
59                 }
60
61                 // The number of ballots argument is not formatted correctly
62                 System.err.println("Invalid number of ballots: \""+args[0]+"\". The
        number of ballots must be at least "+MINIMUM_BALLOTS+".");
63             }
64         }
65     }
66
67     /**
68      * Outputs the generate_ballots correct usage explanation.
69      */
70     private static void display_usage() {
71         System.out.println("USAGE: generate_ballots <number-of-ballots>");
72     }
73
74     /**
75      * Generates the ballots and stores them in the database.
76      * @param num The number of ballots to generate.
77      * @return True if successful, false if there is an error.
78      */
79     private static boolean generate(int num) {

```

```

80     System.out.println("Generating "+num+" ballots...");
81     Connection db = dbconn("ballots");
82     PreparedStatement pst = null;
83     ResultSet rs = null;
84     try {
85         // Open streams
86         File fp = new File(PUBLIC_KEYS_FILE_NAME);
87         FileOutputStream fos = new FileOutputStream(fp);
88         OutputStreamWriter osw = new OutputStreamWriter(fos);
89         BufferedWriter bw = new BufferedWriter(osw);
90
91         // Clear existing ballots from database
92         pst = db.prepareStatement("DELETE FROM allballots");
93         pst.execute();
94         pst.close();
95
96         // Constants for use during ballot generation
97         final BigInteger q = new BigInteger(qAsBase16, 16);
98         final BigInteger g = new BigInteger(gAsBase16, 16);
99         final BigInteger qminusone = q.subtract(BigInteger.ONE);
100
101         // Lists of data
102         List<BigInteger> xis = new ArrayList<BigInteger>();
103         List<BigInteger> yis = new ArrayList<BigInteger>();
104         List<BigInteger> randomPublicKeys = new ArrayList<BigInteger>();
105         List<BigInteger> restructuredPublicKeys = new ArrayList<BigInteger>();
106         List<BigInteger> baseCryptograms = new ArrayList<BigInteger>();
107
108         // Generate xi's and storing g^{xi}
109         for (int i=0; i<num; i++) {
110             BigInteger xi;
111             do {
112                 xi = new BigInteger(q.bitLength(), random);
113             } while(xi.compareTo(qminusone) > 0);
114             xis.add(xi);
115             randomPublicKeys.add(g.modPow(xi, q));
116         }
117
118         // Generate yi's and storing g^{yi}
119         for (int i=0; i<num; i++) {
120             BigInteger yi = BigInteger.ZERO;
121             int j = 0;
122             while(j<i) {
123                 yi = yi.add(xis.get(j));
124                 j++;
125             }
126             j++;
127             while(j<num) {
128                 yi = yi.subtract(xis.get(j));
129                 j++;
130             }
131             yis.add(yi);
132             restructuredPublicKeys.add(g.modPow(yi, q));
133         }
134

```

```

135     // Check the cancellation formula holds
136     BigInteger sum = BigInteger.ZERO;
137     for (int i=0; i<num; i++) {
138         sum = sum.add(xis.get(i).multiply(yis.get(i)));
139     }
140     if (sum.equals(BigInteger.ZERO)) {
141         System.out.println("[YES] Cancellation formula holds.");
142     }
143     else {
144         System.out.println("[ERROR] Cancellation formula does NOT hold.");
145     }
146
147     // Compute the "base" cryptograms:  $g^{\{xiyi\}}$ 
148     for (int i=0; i<num; i++) {
149         BigInteger xi = xis.get(i);
150         BigInteger yi = yis.get(i);
151         BigInteger gexpxiyi = g.modPow(xi.multiply(yi), q);
152         baseCryptograms.add(gexpxiyi);
153     }
154
155     // Add cryptograms to database and output file
156     for (int i=0; i<num; i++) {
157         BigInteger no = baseCryptograms.get(i);
158         BigInteger yes = no.multiply(g.modPow(BigInteger.ONE, q));
159         BigInteger rnd = randomPublicKeys.get(i);
160         BigInteger rst = restructuredPublicKeys.get(i);
161         pst = db.prepareStatement("INSERT INTO allballots (number,
random-public-key, restructured-public-key, no-cryptogram, yes-cryptogram)
VALUES (?, ?, ?, ?, ?)");
162         pst.setInt(1, i+1);
163         pst.setString(2, rnd.toString());
164         pst.setString(3, rst.toString());
165         pst.setString(4, no.toString());
166         pst.setString(5, yes.toString());
167         pst.execute();
168         pst.close();
169
170         // Output to file
171         int ballotNum = i+1;
172         bw.write(ballotNum+" "+rnd.toString()+" "+rst.toString());
173         bw.newLine();
174         bw.flush();
175     }
176
177     // Close streams
178     bw.close();
179     osw.close();
180     fos.close();
181
182     return true;
183 }
184 catch (Exception e) {
185     // Something went wrong...
186     e.printStackTrace();
187     return false;

```

```
188     }
189     finally {
190         // Ensure everything is closed
191         try {
192             if (db != null) {
193                 db.close();
194             }
195             if (pst != null) {
196                 pst.close();
197             }
198             if (rs != null) {
199                 rs.close();
200             }
201         }
202         catch (SQLException e) {
203             e.printStackTrace();
204         }
205     }
206 }
207
208 /**
209  * Gets a database connection.
210  * @param dbName The name of the database to be connected to.
211  * @return A connection to the database if successful, null if not.
212  */
213 private static Connection dbconn(String dbName) {
214     // Database connection
215     Connection con = null;
216
217     // Access details
218     String url = "jdbc:mysql://localhost:3306/"+dbName;
219     String user = "root";
220     String password = "PASSWORD";
221
222     try {
223         // Create connection
224         con = DriverManager.getConnection(url, user, password);
225     } catch (Exception e) {
226         System.err.println(e);
227     }
228
229     // Return the connection to the database
230     return con;
231 }
232 }
```

LISTING A.1: Ballot Generation Source Code

References

- [1] The 2008 New Zealand General Election: Provisional Results. [Online]. Available: <http://www.parliament.nz/en-nz/parl-support/research-papers/00PLLawrp08041/the-2008-new-zealand-general-election-provisional-results>
- [2] The 2011 General Election: Provisional Results. [Online]. Available: <http://www.parliament.nz/en-nz/parl-support/research-papers/00PLLawRP11041/the-2011-general-election-provisional-results>
- [3] Voter and Non-Voter Satisfaction Survey 2011. [Online]. Available: <http://www.elections.org.nz/events/past-events-0/2011-general-election/reports-and-surveys-2011-general-election/voter-and-non>
- [4] Smartphone Penetration in New Zealand. [Online]. Available: <http://think.withgoogle.com/mobileplanet/data/omp-2013-data-nz.xlsx>
- [5] F. Hao, P. Y. Ryan, and P. Zieliński, “Anonymous voting by two-round public discussion,” *IET Information Security*, vol. 4, no. 2, pp. 62–67, 2010.
- [6] F. Hao and M. N. Kreeger, *Every Vote Counts: Ensuring Integrity in Large-Scale DRE-based Electronic Voting*. Newcastle University, Computing Science, 2011.
- [7] F. Hao, B. Randell, and D. Clarke, “Self-enforcing Electronic Voting,” in *Security Protocols XX*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7622, pp. 23–31. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35694-0_4
- [8] A. Kiayias and M. Yung, “Self-tallying Elections and Perfect Ballot Secrecy,” in *Public Key Cryptography*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, vol. 2274, pp. 141–158. [Online]. Available: http://dx.doi.org/10.1007/3-540-45664-3_10
- [9] T. Storer and I. Duncan, “Electronic Voting in the UK: Current Trends in Deployment, Requirements and Technologies.” in *PST*, 2005.

- [10] A. Ansper, S. Heiberg, H. Lipmaa, T. A. Øverland, and F. Van Laenen, "Security and Trust for the Norwegian E-voting Pilot Project E-valg 2011," in *Identity and Privacy in the Internet Age*. Springer, 2009, pp. 207–222.
- [11] D. Clarke, F. Hao, and B. Randell, "Analysis of Issues and Challenges of E-voting in the UK," in *Proceedings of the 20th International Conference on Security Protocols*, ser. SP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 126–135. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35694-0_14
- [12] R. M. Alvarez, T. E. Hall, and A. H. Trechsel, "Internet Voting in Comparative Perspective: The Case of Estonia," *PS: Political Science & Politics*, vol. 42, pp. 497–505, 7 2009. [Online]. Available: http://journals.cambridge.org/article_S1049096509090787
- [13] J. Gerlach and U. Gasser, "Three case studies from switzerland: E-voting," *Berkman Center Research Publication No*, vol. 3, 2009.
- [14] S. Wolchok, E. Wustrow, D. Isabel, and J. A. Halderman, "Attacking the washington, DC Internet voting system," in *Financial Cryptography and Data Security*. Springer, 2012, pp. 114–128.
- [15] W. Rankl and W. Effing, *Smart card handbook*. John Wiley & Sons, 2010.
- [16] H.-Y. Chien, J.-K. Jan, and Y.-M. Tseng, "An efficient and practical solution to remote authentication: smart card," *Computers & Security*, vol. 21, no. 4, pp. 372–375, 2002.
- [17] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
- [18] C. Lim, Y. Dan, K. Lau, and K. Choo, "Smart card reader," *Consumer Electronics, IEEE Transactions on*, vol. 39, no. 1, pp. 6–12, 1993.
- [19] K. Weldemariam and A. Villafiorita, "A Survey: Electronic Voting Development and Trends," 2010.
- [20] C.-I. Fan and W.-Z. Sun, "An efficient multi-receipt mechanism for uncoercible anonymous electronic voting," *Mathematical and Computer Modelling*, vol. 48, no. 9-10, pp. 1611–1627, 2008. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0895717708001787>
- [21] D. A. Gritzalis, "Principles and requirements for a secure e-voting system," *Computers & Security*, vol. 21, no. 6, pp. 539–556, 2002.

- [22] W.-C. Ku and S.-D. Wang, “A secure and practical electronic voting scheme,” *Computer Communications*, vol. 22, no. 3, pp. 279–286, 1999.
- [23] H.-T. Liaw, “A secure electronic voting protocol for general elections,” *Computers & Security*, vol. 23, no. 2, pp. 107–119, 2004.
- [24] H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*. SIAM, 1992, vol. 63.
- [25] D. Boneh, “The Decision Diffie-Hellman problem,” in *Algorithmic Number Theory*, ser. Lecture Notes in Computer Science, J. Buhler, Ed. Springer Berlin Heidelberg, 1998, vol. 1423, pp. 48–63. [Online]. Available: <http://dx.doi.org/10.1007/BFb0054851>
- [26] L. N. Childs, “Cyclic Groups and Cryptography,” in *A Concrete Introduction to Higher Algebra*, ser. Undergraduate Texts in Mathematics. Springer New York, 2009, pp. 387–412. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-74725-5_19
- [27] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung, “Multi-Authority Secret-Ballot Elections with Linear Work,” in *Advances in Cryptology - EUROCRYPT '96*, ser. Lecture Notes in Computer Science, U. Maurer, Ed. Springer Berlin Heidelberg, 1996, vol. 1070, pp. 72–83. [Online]. Available: http://dx.doi.org/10.1007/3-540-68339-9_7
- [28] R. Cramer, R. Gennaro, and B. Schoenmakers, “A secure and optimally efficient multi-authority election scheme,” *European transactions on Telecommunications*, vol. 8, no. 5, pp. 481–490, 1997.
- [29] B. Adida, O. De Marneffe, O. Pereira, and J.-J. Quisquater, “Electing a University President Using Open-audit Voting: Analysis of Real-world Use of Helios,” in *Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, ser. EVT/WOTE'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 10–10. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1855491.1855501>
- [30] E. Rescorla, *SSL and TLS: designing and building secure systems*. Addison-Wesley Reading, 2001, vol. 1.
- [31] J. Benaloh, “Rethinking Voter Coercion: The Realities Imposed by Technology,” in *Presented as part of the 2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections*. Berkeley, CA: USENIX, 2013. [Online]. Available: <https://www.usenix.org/conference/ewtwote13/workshop-program/presentation/Benaloh>

- [32] F. Zagórski, R. T. Carback, D. Chaum, J. Clark, A. Essex, and P. L. Vora, “Remote integrity: Design and use of an end-to-end verifiable remote voting system,” in *Applied Cryptography and Network Security*. Springer, 2013, pp. 441–457.
- [33] W.-M. Lee, *Beginning android 4 application Development*. John Wiley & Sons, 2012.
- [34] Spongy Castle by rtyley. [Online]. Available: <http://rtyley.github.io/spongycastle/>
- [35] The Legion of the Bouncycastle. [Online]. Available: <https://www.bouncycastle.org/>
- [36] TrustyTahr/ReleaseNotes - Ubuntu Wiki. Oracle Corporation. [Online]. Available: <https://wiki.ubuntu.com/TrustyTahr/ReleaseNotes>
- [37] R. Petersen, *Ubuntu 14.04 LTS Desktop: Applications and Administration*. Surfing Turtle Press, 2014.
- [38] MySQL 5.5 Release Notes. Oracle Corporation. [Online]. Available: <http://dev.mysql.com/doc/relnotes/mysql/5.5/en/news-5-5-37.html>
- [39] Java SE 7 Update Release Notes. Oracle Corporation. [Online]. Available: <http://www.oracle.com/technetwork/java/javase/7u-relnotes-515228.html>
- [40] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley, *The Java virtual machine specification*. Pearson Education, 2014.
- [41] JavaMail 1.4.5 Release Notes. Oracle Corporation. [Online]. Available: <http://www.oracle.com/technetwork/java/javamail145notes-1562619.html>
- [42] RFC 6979 - Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). The Internet Engineering Task Force. [Online]. Available: <http://tools.ietf.org/html/rfc6979#appendix-A.2.2>
- [43] SSLSocket (Java Platform SE 7). Oracle Corporation. [Online]. Available: <http://docs.oracle.com/javase/7/docs/api/javax/net/ssl/SSLSocket.html>
- [44] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, “An architecture for a secure service discovery service,” in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM, 1999, pp. 24–35.
- [45] TrustManager (Java Platform SE 7). Oracle Corporation. [Online]. Available: <http://docs.oracle.com/javase/7/docs/api/javax/net/ssl/TrustManager.html>

- [46] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard, “Practical Multi-candidate Election System,” in *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '01. New York, NY, USA: ACM, 2001, pp. 274–283. [Online]. Available: <http://doi.acm.org/10.1145/383962.384044>