COERCION-RESISTANT VERIFIABLE WEB-BASED ELECTIONS IN LINEAR TIME

by

Mir Masood Ali

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2019

This thesis is dedicated to my parents, both of whom have made huge sacrifices to get me here.
Pappa's been my biggest inspiration. He has always driven me forward, to achieve and to excel.
Mummy's the most active person I know. She's at it every day, come hail or storm. Her unconditional love always makes me feel like I belong.
I love you, and I hope that with my first shot at research, I've made you proud.

Table of Contents

List of Tables

List of Figures

Abstract

Voter coercion broadly includes any attempt to influence the secrecy of a voter's ballot, including bribery, vote stealing, and forced abstention. This risk, which affects the fundamentals of privacy in a democracy, is especially prevalent in remote voting systems. Numerous coercion-resistant protocols have been proposed that manage to counter this risk, but compromise on time-complexity and practicality.

This thesis proposes a new protocol for a coercion-resistant, verifiable remote voting system that tallies votes in linear time. The protocol is modeled using observational equivalences in Applied Pi Calculus. This model is then verified in ProVerif to determine its compliance with a formal definition of coercion-resistance.

The thesis finally presents a web-based application built using the Django framework. The application implements an election system based on the verified protocol. It provides a user interface to interact with the system and cast votes, while server-side Python scripts create, manage, and tally the election.

The protocol is compared against previous proposals based on the number of modular exponentiations required for a single run of an election. The application is evaluated based on the time taken for the execution of different phases in an election.

The results will lead to implementations of coercion-resistant voting applications that can be deployed in elections where the threat of coercion is considered significant.

List of Abbreviations and Symbols Used

K_{TT}	Election Public Key / Public Key of Tabulation Tellers
K_{V_A}	Voter's Authentication Key
K_{V_D}	Voter's Designation Key
$\mathbf{Z}_q^*/\mathcal{M}$	Message space for plaintext messages, the q order subgroup of the finite field $\mathbf{Z}_p^*$
$\mathcal{N}$	Space of nonces
$\mathcal{O}$	Universal integer space for witnesses
$\mathcal{O}(N)$	Worst-case complexity of an algorithm
$\mathcal{P}$	Prover of NIZK
$\mathcal{V}$	Verifier of NIZK
σ_i/s_i	Credential share
$\textsf{Enc}(v, r)$	Encryption of value v under randomness r
$\textsf{Enc}_\epsilon(v)$	Encryption of value v using public key ϵ
$\textsf{PET}(X, Y)$	Plaintext Equivalence Test
$c = (\alpha, \beta)$	ElGamal Ciphertext
g	Generator of Finite Field, $\mathbf{Z}_p^*$
NIZK	Non-Interactive Zero Knowledge Proof

Acknowledgements

This Masters thesis was my introduction to research. It's a field and an approach that was alien to me just a year and a half ago. None of this would have been possible had I not received incredible amounts of support, guidance, attention, and care from my supervisor, Dr. Srinivas (Srini) Sampalli.

I will forever be grateful to Srini for taking a chance with me and giving me feedback with both, encouragement and criticism, in equal amounts. He's taught me to create healthy collaborations and productive teams, and successfully sustain good relationships with my peers. Srini's lab was my home away from home. I shall try to replicate his ideas of a healthy workspace, his zest for knowledge, and his love for teaching, everywhere I go.

I would like to thank my readers, Dr. Michael McAllister and Dr. Nur Zincir-Heywood, and the chair for my defense, Dr. Qiang Ye for agreeing to sit on the committee for my thesis. Their questions, inputs, and suggestions contributed towards making this thesis better.

Dr. Michael McAllister helped me navigate algorithms that refused to make sense to me. He additionally made me realize that this Masters thesis cannot solve all of the problems in the world. He's been immensely sweet and encouraging, making time to hear me out despite numerous administrative responsibilities.

This thesis would not have been possible without oodles of hugs, love, and support from Robbie MacGregor. Da, as I call him, has been unimaginably patient in listening to me rant and calming me down countless times. He's been my nicest friend and my most encouraging peer in the lab. We make a great team. I'll forever cherish my interactions with him.

Moustafa (Steve) Dafer and his inquisitiveness have both played separate, but significant parts in the completion of this thesis. He's always been ready to provide insightful conversation. Steve's continuous showering of questions helped me figure the 'Linear Time' part of the system. He's an ocean of knowledge, but (un)surprisingly, his questions always do the trick.

Dr. Hiroyuki Ohno was my first collaboration as a part of the lab. I met him at a time when I was yet to find my research question. His refusal to sleep and his love for his devices make him an absolute delight to work with. I aspire to be as energetic as him every single day, and I expectedly fall short.

Dr. Kirstie Hawkey taught me to look at research differently than one would look at projects in general. The two courses that I took with her made for some of the most helpful sessions that I've sat through. She shaped the way I approach papers and lent an ear every time I felt like I needed direction.

I also received support and guidance from Dr. Raghav Sampangi when I was beginning to shape my research. Regardless of the topic of the conversation, he's a delight to talk to.

I can't thank Dr. Sudeeksha Subramanya enough for tolerating me and my anxiety. I wouldn't have been able to be there for anyone like she has been there for me. Words fall short here, and I shall let them.

My time at Dal wouldn't have been half as lovely had it not been for friends like Oyshee Saha Roy, Aditi Nair, and Reetam Taj. They are my biggest pillars of support. Had they not coerced me to join them, I would not have been spotted outside the lab.

Dinesh Shenoy is ready for jokes as much as he is up for serious talk. The best food in Halifax is prepared at his home. Had I gone the industry route, I'd want him to manage my team.

Sagarika Ghosh listens to the advice that I give her, when it really should be the other way around. She's going to be amazing PhD student and I'm calling dibs on front-row seats for this drama fest.

My peers at MyTech Lab are the healthiest, happiest bunch of researchers. Amjad Alsirhani, Hisham Allahem, Bader Aldughayfiq, Junhong Li, Dr. Saurabh Dey, and Pat Crysdale provide the most entertaining conversations and are ever ready to pitch in and help. My friends from home, Adithya and Vignesh, are always happy to support me.

My brother, Mudassir, is always up and about taking things one notch higher. With this thesis and the next, that's exactly what I'll be trying to do. I love you.

Chapter 1

Introduction

Researchers have, in the last three decades, covered long strides in the ability of personal computers to execute cryptographic protocols in a manner that was previously presumed infeasible. Remote election systems can harness this to make voting applications available, verifiable, and accessible. As a result, remote and Internet voting has been adopted for federal elections in countries like Estonia [2, 3] and South Africa [4], while in-person electronic voting is used in Brazil [5] and India [6], among others. The security of these systems, however, remains a concern.

According to a 2017 report from Canada's Communications Security Establishment (CSE) [7], potential attacks on elections against Canada's democratic process can successfully prevent citizens from registering to vote in elections, prevent registered voters from casting their vote, tamper with election results, or steal the voter database. The same report also states that 13% of all countries that held federal elections in 2017 faced an attack on their voting process.

Recommendations to the US government regarding best practices to record voter intention can be found in *Securing the Vote* [8]. According to the report, there was evidence regarding the spread of misinformation and intrusion of voter databases in the 2016 U.S. General Elections. Additionally, despite the lack of evidence regarding attacks on vote casting and counting as stated in the report, the absence of verifiability ensures that demands for recounts recur in future elections.

Presidential candidates in the U.S. General Elections have made multiple allegations against questionable tallies. The Florida Recount during the U.S. General Elections in 2000 [9] and more recently, the demands for recount in 2016 [10] serve as prime examples of distrust in unverifiable counting. Cryptographic voting helps overcome this flaw in current voting systems by enforcing user trust through auditable tallying.

Coercion is a known problem in university elections for student unions. Allegations and accusations aside from physical coercion have been cited despite the use of trusted paper ballots [11].

There is a need for countering voter influence, intrusion in voter registration databases, and speculation of mishandled tallying. Remote cryptographic protocols can be implemented as web-based applications not only to make them verifiable and coercion resistant, but also easily accessible.

1.1 The Voting Problem

The voting problem refers to the issue around providing ballot secrecy to voters, ensuring that the votes that they cast remain a secret, while also providing them the ability to verify that their vote has been correctly recorded and tallied.

This often leads to the notion of a receipt, which a voter can use to verify the inclusion of their vote in the tally. Most voting protocols rely on the notion of receipt-freeness. The concept refers to the absence of a plaintext receipt (one that displays the marked ballot in plaintext) that a voter could be forced produce to demonstrate that they cast their vote as intended by an adversary.

1.2 Coercion-Resistant Elections

Juels et al. [12] were the first to identify the inadequacy of receipt-freeness. The definition was expanded to include the potential of randomizing a ballot, making a voter abstain from voting, and simulating a vote on behalf of the voter. They proposed the use of a protocol to resist coercion by generating fake credentials. Often referred to as the JCJ Protocol, one of its primary criticisms is the complexity of duplication and invalid vote removal, which made it hard to implement. A version of the JCJ Protocol, modified to cast votes into manageable ballot boxes made it to a reasonable implementation as Civitas in Java and Jif [13] by Clarkson et al. [14].

Enhancements were suggested by Smith and Weber [15, 16, 17] regarding the use of a shared random value to generate blinded encryptions that are added to a hash table for collision detection. This leaves voter choices exposed by revealing the value of a known choice obfuscation.

Similarly, Araujo et al. [18, 19, 20] suggest increasing the efficiency by using group signatures. However, in this case, coerced registration tellers have the ability to introduce incorrect credential shares.

The suggestions made by Spycher et al. [21] provide for simple practical implementation without introducing errors that may have been possible in previous suggestions. The use of indices makes duplicate and invalid vote elimination achievable without compromising on the coercion resistance of the system.

1.3 Contribution of this Thesis

This thesis presents Resist, a coercion resistant election protocol that uses the suggestions made by Spycher et al. [21] to the JCJ Protocol [12]. The suggested protocol reduces the complexity of tallying votes in coercion-resistant voting systems down to linear time (in the number of votes cast). The protocol is modeled and verified to ensure that it does not compromise on the security assumptions that were satisfied by the JCJ Protocol.

1.4 Outline of the Thesis

The rest of the thesis is structured as follows. Chapter 2 explores the evolution of voting systems, the problems associated with them, and a few popular voting implementations. Chapter 3 introduces the concept of coercion resistance, the progress made in the field so far, and the research gap that we are attempting to address.

The Protocol behind Resist is introduced in Chapter 4. The players, entities, and phases are expanded upon. In Chapter 8 models the proposed protocol in terms of observational equivalences. The model is fed into ProVerif and the results are discussed against a formal definition of Coercion Resistance.

The validated protocol is expanded upon in terms of the algorithms for encryption and proofs, and accompanied by associated protocols in Chapter 6. These algorithms are adapted to code in Python and Javascript into a web-based application in Chapter 7.

The protocol and implementation are evaluated and compared against previously proposed protocols in Chapter 8. The limitations of the system and possible extensions to the system are discussed in Chapter 9, where a conclusion is provided.

Chapter 2

Background

2.1 A Peek Into Democracy and Voting

Elections, voting, and democracy can be traced as far back as 508 B.C. to Athens, where male land owners voted for the exile of political candidates[22]. The ballots for these elections were pieces of pot on which choices were written. A total of 6000 votes meant that the candidate in question was sent for an exile.

The Greek introduced democracy to the world, which took centuries to witness growth and adoption. It wasn't until the turn of the 13th century that a positive form called approval voting, was adopted by medieval Venice. It was the modern form of participating in a democracy. Each voter would cast a single vote for the candidate of their choice and none for all other candidates. The candidate with the highest number of votes at the end was declared the winner of the election [23].

Elections in the United States were introduced without the concept of ballot secrecy. The act of oath taking and casting a vote was majorly performed in broad public view. Multiple clerks were employed, who would record these votes and compare them to flag errors. This concept of voting in viva-voce found its way into George Caleb's "The County Election", shown in Figure 2.1.

Canada adopted elections from the beginning of the Confederacy in 1867. As the country adopted democracy, it initially allowed for male property owners and limited to certain regulations. Women's right to vote in federal elections wasn't introduced until 1918. First Nation and Indigenous citizens had to give up their special status to participate in elections prior to 1960. The Canadian Charter of Rights and Freedoms, introduced in 1982, granted all Canadian citizens, above the age of 18, with the right to participate in elections [25, 26, 27].

In 1858, Victoria in South Australia printed passed an act that changed the way elections were conducted [28]. Ballots were printed on paper and stored securely until the day of the election. On the day of the election, one ballot was handed out to each

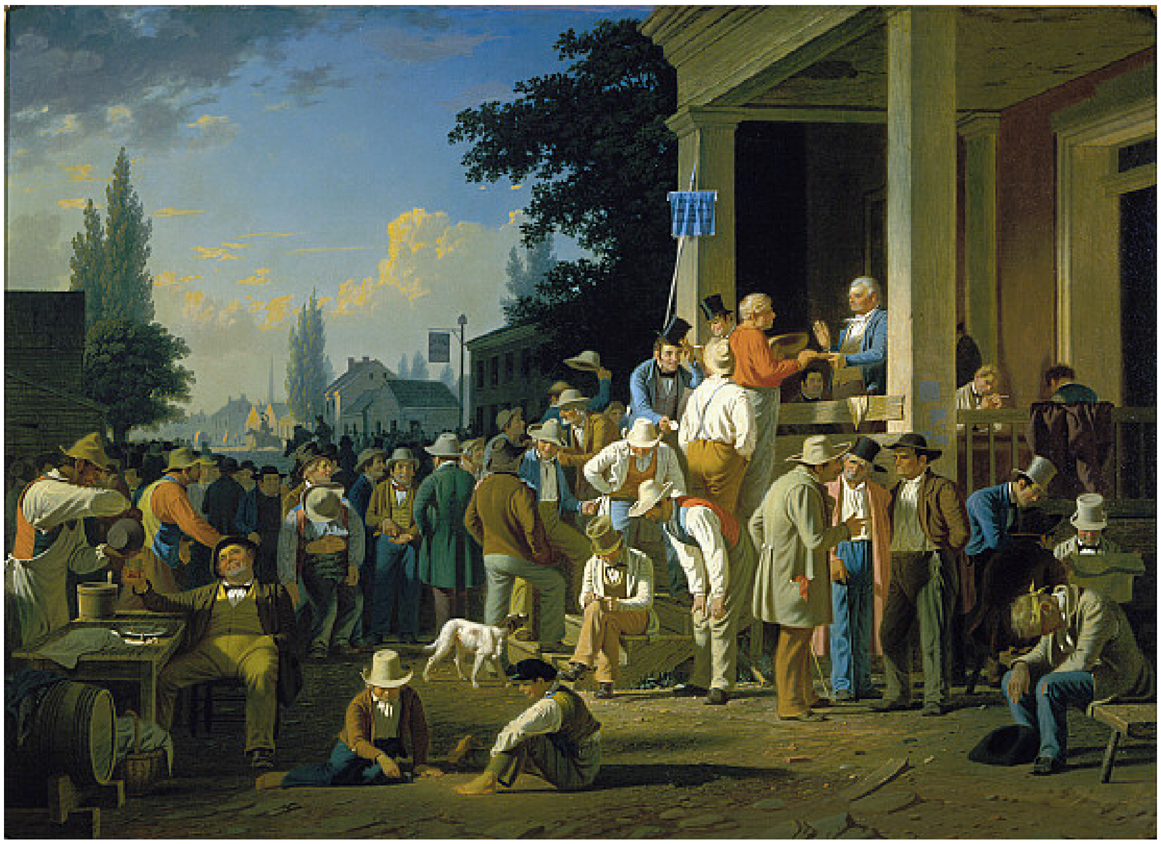


Figure 2.1: George Caleb [24], popularly referred to as the “Missouri artist”, painted this image in 1852. Titled “The County Election”, the painting displays a man taking oath at the entrance of the courtroom. He would then proceed to announce his vote for the election while two clerks (seated behind the judge), took note. The entire process takes place in the midst of a bustling market place, where the threat of coercion is high. The image is often cited to represent the absence of the notion of privacy during elections.

voter. The voter then proceeded to fill the ballot in booths that provided privacy and isolation. The ballots were then folded up and dropped into a closed box, which would only be opened at the time of tallying.

The Australian ballot (or the secret ballot), has since been widely adopted as the de facto voting standard. It was introduced in Canada in 1874 and the United States in 1888.

The provision of paper ballots led to the introduction of mechanized recording of votes. New York introduced the concept of pulling levers to record candidate choices in 1892 (some of which are used even today) [29], while the punch card system was introduced in the 1960s [30]. In 1965, when Joseph Harris introduced Votomatic, elections throughout the United States shifted from paper ballots to mechanized voting systems [31]. Despite recorded irregularities and discrepancies in the manufacture, design, and implementation of punch cards continue to be used in elections to this day [32].

Computerization of voting began in the 1960s, with the introduction of optical scanners. An deeper look into this is provided in Section 2.3.

2.2 What Makes Voting Hard?

The voting problem is a complex problem that is yet to find answers. The participants in the system cannot trust each other, and the only way that elections can be conducted in a smooth, friction-less manner is if the players in the system trust the system itself.

This section of the Background amalgamates the description of the voting problem from explanations by Ben Adida [33] and Bernhard et al. [34].

2.2.1 Lack of Trust

In any election that faces significant adversarial threat, election officials, voters, and all players that interact with the system do not trust each other. If Alice, an eligible voter, casts a vote using the election system, they need confidence in the proper functioning of the system. The system should be in a position to prove to Alice, when challenged, that their vote was recorded as intended.

Most cryptographic voting systems include a public bulletin board that can even be accessed by external entities that do not participate in the votng of the system. The bulletin board usually includes votes that have been cast (either in plaintext or in an encrypted form), proofs of proper functioning (encryption, verification, and tallying) of election officials, and final results of the election.

2.2.2 Evidence and Receipts

Proving proper recording and inclusion of votes in the system is a given in trustworthy systems. They can either display the votes in plaintext or a verifiable form.

On successfully casting a vote, Alice should receive a receipt that either includes the ballot with the marked choices, a summary of their filled ballot, or an encryption/hash that they can use for verification.

2.2.3 Secrecy

Since the introduction of the secret ballot in 1858, there has hardly been a voting system that has proposed linking a voter to the contents of their vote. Most voting systems and voters interacting with it favour the secrecy of voter choices for the smooth functioning of a democratic institution.

Secrecy, however, comes in the way of the aforementioned verifiability of an election system. To explain this, we introduce Carl, a new player in the election system. Carl intends to make Alice vote for a certain candidate. They demand that Alice reveal to them the contents of their ballot so that Alice is forced to comply.

Keeping the ballot secret can help Alice keep the contents from Carl. Secrecy can be guaranteed by ensuring that ballots are marked in isolation (as is done with paper ballots in polling booths) or by altering the structure of the ballot itself (as is done by encrypting votes in remote elections).

Guaranteeing secrecy from an adversary like Carl, unfortunately also entails secrecy from Alice. This flies in the face of verifiability, wherein the system needs to assure Alice that their vote had been cast as intended. Therefore, the receipt that Alice receives for verification can only serve as proof to Alice and not Carl. This is the crux of the voting problem.

2.2.4 Auditable Tallying

After votes have been tallied and election results have been computed, a verification of the tally can be performed using statistical audits. In such audits, random samples of different sizes (that vary by method) of the trail are selected and computed. Statistical audits detect errors in the computation and give the assurance, with a reasonable margin of error, that the outcome of the election is correct. The Democratic Candidate for the 2020 Presidential Election, Elizabeth Warren, has demanded making Risk Limiting Audits mandatory for elections across the United States [35].

Bernhard et. al. [34] state that an audit is said to be a risk limiting audit (RLA), given a risk limit of α, if when the reported outcome of the audit is incorrect, there is a probability of at least $1 - \alpha$ that the audit will lead to correcting the outcome.

2.3 Technology and Elections

2.3.1 Optical Scanners

The Norden Electronic Vote Tallying System introduced optical scanners to record secret ballots for tallying was one of the first optical scanners deployed in elections in the United States [36]. The Mark Sense system [37], introduced in 1965, included a photosensor that would scan ballots and identify any mark that appeared darker than the others.

Optical scanners simplify the recording of votes. They were used to reduce the strain of individually counting votes by hand, which often contributed to a rise in human error [38]. It also introduces attacks in the system wherein a voter can obtain an unmarked ballot and record votes on the voter's behalf.

Optical scanners continue to be used in voting systems today [39]. They are usually deployed in association with a ballot marking device. Security and trust implications of using an optical scanner have long been discussed and are often countered using verifiability and RLA's.

2.3.2 Direct Record Electronic Voting Machine

A Direct Record Electronic(DRE) Voting Machine is the equivalent of marking a ballot via a display shown to the voter. The record can be taken via a mechanical button or a touch screen. The vote is recorded with the help of a program stored on the machine itself.

DRE's are used in elections across Brazil [5], India [6], and Malaysia [40]. They had previously been deployed in Germany, Netherlands, Ireland,and the United Kingdom, but had been called off for being unreliable [41, 42, 43, 44].

DRE's have been used in federal elections because they provide options to display the ballot in multiple languages; they can be used along with headphones for voters with accessibility issues; and they simplify ballots by reducing them from multiple pages required for each paper ballot [45].

DRE's are often supplied by commercial manufacturers that keep the source code closed, leading to the inability to verify if the system meets the security requirements that election systems require [46].

2.3.3 Ballot Marking Devices

A specific type of DRE is a Ballot Marking Device (BMD). It helps a voter record their choices on a physical ballots. Unlike conventional DRE's, they do not store the ballot or scan them. While providing the same accessibility options provided by a DRE, a BMD does not include the vulnerability introduced by having to store the vote [47].

A BMD is often used to print out ballots that are later scanned by a precinct-level or central optical scanner. This was introduced to counter incorrect scans resulting from hand-marked ballots [9].

2.3.4 Internet and Remote Voting

Countries around the world have attempted to debate the perks of using the Internet as a means of remote voting. The system has obvious ease and accessibility advantages associated with it, but it introduces new vulnerabilities, primarily those of coercion and eavesdropping [7].

Internet voting has long been debated as a means of increasing voter turnout, a claim debunked by a study in two Swiss elections [48]. Despite this, countries like Estonia [2, 3] and South Africa [4] continue to use the Internet to conduct elections.

Given an increase in the capabilities of regular systems to handle cryptographically secure votes, coupled with verifiable proofs at every step, Internet voting continues to be considered as a method of voting that is to be aspired in the event that systems manage to satisfy the identified caveats of a paper-based voting system [34].

Chapter 3

Literature Survey

3.1 Internet Voting Schemes and Protocols

In this section, we survey a few Internet Voting Systems. This list is not comprehensive, but instead has been provided to cover the attempts at implementation made so far.

3.1.1 Selene

Developed in the University of Luxembourg, Selene [49] offers a simplistic and more practical approach toward handling cryptographic voting. The purpose of a scheme like Selene is the implication that the general public, one without the necessary technical prowess to adjudge the working of cryptographic voting. Certain federal governments, in particular the German Federal Law, disallow the use of cryptography in voting, citing the inability of voters to understand the mechanisms behind it.

Selene instead offers to keep votes in plaintext on a public bulletin board. These votes are, however, associated with a ballot tracking number assigned to a voter. Selene ensures that ballot tracking numbers are unique and voters are made aware of this only at the end of a voting phase. As a result of posting these votes in plaintext, voters do not have to attempt a complicated Benaloh Challenge to verify their encryption [50].

This obscures the identity of the voter from the general public, and especially from potential coercers. A voter can instead claim any other coercion-compliant vote to be theirs with no way for the coercer to prove otherwise.

Selene initially offered mitigation instead of resistance, with the possibility of failure majorly resting on a coercer being a voter themselves. This was later mitigated with a proposal in approach that the authors chose to address in Selene II, providing a Civitas' like diversion, with uniquely allotted deceptive ballot trackers.

3.1.2 Civitas

A Cornell University implementation of the JCJ Protocol [12], Civitas [14] is a system for remote voting developed in Java and a security-typed extension of Java, Jif. The implementation slightly modifies the casting and tallying process proposed in JCJ, in order to make them manageable and practical.

Civitas involves an initial physical phase (or an untappable channel) for obtaining keys. These keys then allow voters to obtain election-specific credentials remotely for any number of elections thereafter.

Civitas assumes the trustworthiness of the election supervisor that adds a voter and the registrar that provides initial keys. The registration tellers and tabulation tellers are thereafter required to provide proof of correct registration and tallying respectively.

Voters obtain shares of their election-specific credentials from multiple registration tellers, the product of which builds the credential that validates their vote. Each of these shares are associated with a Designated-Verifier Reencryption Proof (DVRP) based on Hirt and Sako [51]. In the event of coercion, the voter alters one of the shares of their credential and generates a modified DVRP, in order to create an indistinguishable fake credential. A coercer can then cast a vote with a fake credential with no way of verifying if it was included in the final tally.

Civitas does not assume the robustness of a single bulletin board, and hence allows a user to cast their vote to multiple ballot boxes, thus leaving it to the tallier to eliminate duplicates. Civitas also does not have a front-end implementation with a graphical user interface (GUI).

The protocol behind Civitas is explained in greater detail in Section 3.4.

3.1.3 Helios

Helios [52] is an open audit voting system, primarily made for simple access through web browsers. Developed for low-coercion elections, the system is recommended for implementations that do not foresee potential adversarial influence on ballot privacy.

The system was made with the intention of making simple elections verifiable and open audit. Helios lays emphasis on the ability of a voter to verify their encryption (using the Benaloh challenge [50], observe their vote on the public bulletin board (with no attempt at hiding voter identity), and the ability of any individual to audit any part of the election. Each phase of the election, including voter registration, casting, and tallying, are associated with non-interactive zero-knowledge proofs.

Helios has been used in university elections [53] and has been made available online for anyone to create and run elections without the need to implement their own personal servers.

3.1.4 Prêt à Voter

After finishing second in the University Voting Systems Competition, 2007, the team from the University of Surrey [54] completed an EPSRC-funded verifiable implementation of Chaum [55]. Keeping end-to-end (E2E) verifiability as its primary motivation, the system implements cryptographic voting along with candidate order randomization.

One of the first attempts at providing audibility of the final tally, while also dissociating voters from their votes. Despite this dissociation, the receipt that the voters receive includes a privacy protected receipt. Candidate randomization helps hide the voter's intent on this receipt.

Pret a Voter additionally supports Single Transferable Votes (STV), Human Readable Paper Audit Trails (HRPAT), and simultaneous remote and physical voting. The system is modelled as a physical election system that supports electronic and online elections.

3.1.5 Scantegrity II

Prêt à Voter came second to Punchscan [32], an optical scan voting system at the University Voting Systems Competition [56]. The system, has since evolved into Scantegrity, the second version of which was employed in the municipal elections at Takoma Park, Maryland, in 2009 and 2011. The system, designed as an open source Java implementation, requires precincts and optical scanners.

Credited to the contributions of cryptographers David Chaum and Ronald Rivest, Scantegrity II is an E2E verifiable voting system that uses invisible ink for confirmation codes that have been printed on ballots. Voters reveal confirmation codes by using a special pen for voting, one that develops the invisible ink. These codes help voters verify that their intended choices have been recorded correctly, without having to reveal this intent in the process. Meanwhile, the tally can be audited to have been correctly computed.

3.2 Extending Receipt Freeness

Juels et al. [12] coined the term coercion resistance in 2005. They were looking to expand on the more traditional notion of receipt freeness that drove a number of voting protocols prior to their proposal.

While receipt freeness is a necessary step towards solving the existing voting problem, its definition isn't broad enough to envelope all forms of coercion. They extended its definition to cover the following attacks.

- **Randomization Attack.** An attacker might coerce a voter to randomize their ballot in such a way that even the attacker and the voter remain unaware of what the ballot comprised. While this may not be directly beneficial to the adversary's or a candidate's interests, it can be detrimental to the expected result of an election.
- **Forced Abstention Attack.** An attacker may force a voter to abstain from voting.
- **Simulation Attack.** After completing registration, an attacker may coerce a voter to divulge their keys/credentials that validate their vote. They can use these to cast a vote on behalf of the voter thereafter.

3.3 The JCJ Protocol

At a time where most receipt-free protocols proposed to solve receipt-freeness by using "untappable" channels, Juels et. al. [12] where the first to present a remote, receipt-free protocol that allowed voters to cast their votes via a tappable channel. The only caveat was that these voters receive their keys (for multiple elections) through a one-time use of an untappable (or in-person) channel.

The protocol that they had proposed in 2005 has since formed the basis for most remote, coercion resistant implementations. Since it has often been referred to using an abbreviation of the authors of the initial protocol, Juels, Catalano, and Jakobsson, the protocol is henceforth referred to (for the remainder of the thesis) as the JCJ Protocol.

The protocol itself has briefly been described below. Note that the terms, coercer and adversary, are used alternatively in this explanation.

- **Registration.** The 'registrars' jointly compute credential σ for a voter. They then encrypt this value with the public key of the talliers, ϵ and cast it to the bulletin board under randomness α_S as $\mathsf{Enc}_\epsilon(\sigma; \alpha_S)$. No single registration teller or voter knows the value of α_S and only the voter knows the value of σ. This protocol works on the assumption that a majority of the registration tellers are trustworthy.

- **Casting a Vote.** When a voter intends to participate in the election, they choose their candidate, c, from the list of candidates, $\mathcal{C}$, on the bulletin board, such that $c \in \mathcal{C}$. They then proceed to encrypt their ballot by casting their vote as a tuple, (A, B), where $A = \mathsf{Enc}_\epsilon(\sigma, \alpha_A)$ and $B = \mathsf{Enc}_\epsilon(c, \alpha_B)$. Each ballot needs to be accompanied by two Non-interactive zero knowledge proofs (NIZK).

  - One NIZK is to prove knowledge of the plaintext of σ since an adversary can re-encrypt the credential from the bulletin board.

  - The second NIZK is to prove that the vote that has been cast is for a candidate from the given list of candidates, $c \in \mathcal{C}$. This prevents randomization attacks.

  If a voter is coerced during this phase of the election, they need just enough time to cast their own vote without the coercer's knowledge. They can then cast a vote on the coercer's behalf using a fake credential, σ' from the same finite field (explained in later chapters). Their ballot can be marked in a similar manner to the one explained above and they can reveal the plaintexts of A and B to the coercer. Since α_S remains hidden from the all parties, the coercer cannot correctly determine the correctness of the credential.

  The voter can alternatively pass σ' to the coercer to let them cast a vote on behalf of the voter.

- **Tallying.** At the end of the Voting Phase, the bulletin board is populated with N votes. Not all of these N votes are valid since the system allows for duplicate and invalid votes to be cast. Before the talliers can sum the votes, they need to eliminate votes in the following order.

  – **Eliminate Invalid Proofs.** First, they verify the NIZK proofs associated with each vote. Each vote has two proofs associated with it. Any vote that has been cast for a candidate from outside the list, $c \notin \mathcal{C}$, is eliminated. Any vote that additionally fails to prove knowledge of σ is discarded as well.

    This elimination requires traversing the list of N votes once, and hence has a linear complexity, $\mathcal{O}(N)$. At the end of the first round of elimination, the invalid votes that remain are either votes cast with invalid/fake credentials or valid votes that have been cast multiple times (duplicates).

    The steps that follow depend on performing a Plaintext Equivalence Test (PET). While the algorithm for doing so is explained in later chapters, the notion is described here. Given two encrypted values, $X = (x, \alpha_X)$ and $Y = (y, \alpha_Y)$, the result of $\mathsf{PET}(X, Y)$ is a boolean value. It returns *true* if the plaintext values are equal, i.e., $x == y$, and it returns *false* otherwise.

  – **Eliminate Duplicates.** Each vote cast on the bulletin board (A_i, B_i) is iterated over and compared with every other vote on the board (A_j, B_j), where $i \neq j$. If $\mathsf{PET}(A_i, A_j)$ returns *true*, either A_i or A_j is eliminated.

    The choice for this elimination depends on a rule created at the start of the election regarding whether the first cast vote is counted or the last one is. This elimination traverses a list of N votes exhaustively, resulting in a complexity of $\mathcal{O}(N^2)$.

  – **Eliminate Invalid Votes.** Every vote, (A_i, B_i) remaining on the bulletin board needs to be compared with all possible credentials, σ_j the list, S, of valid credentials on th bulletin board. Eliminating votes according to the result of $\mathsf{PET}(A_i, \sigma_j)$ can accomplish this but it presents the danger of revealing the invalidity of certain credentials (of specific votes) to the coercer.

The votes are hence shuffled and reencrypted with the help of a re-encryption mixnet (explained in later chapters), resulting in each vote appearing different $(\hat{A}_i, \hat{B}_i)$. The list S of valid credentials is similarly passed through the mixnet, resulting in the re-encryption of every credential σ_j to $\hat{\sigma_j}$.

$\mathsf{PET}(\hat{A}_i, \hat{\sigma_j})$ is performed for every combination of a vote and credential. Any vote that results in $false$ for the test is determined to have invalid credentials and eliminated from the final tally.

Considering that the list S has a size of n and the list of votes has a size of N, the complexity of this step will be $\mathcal{O}(N^2)$. As stated in Civitas [14] and by Spycher et al. [21], if the size of N is way more than n, i.e., if the number of votes cast are greater than n by orders of magnitude, the worse case complexity of the JCJ Protocol turns out to be $\mathcal{O}(N^2)$.

Juels et al. [12] accompanied their protocol by a formal definition of Coercion Resistance. We skip expanding on this and direct the reader to [12] for a primitive version of the notion [12]. A formalization of the term is explained in Chapter 5, where a definition was conceptualized after consideration of further factors and scenarios.

3.4 Civitas

The Civitas system, developed by Clarkson et al. [14], was one of the first attempts at implementing a coercion resistant election system. It was written in Java and Jif, and was based on the JCJ Protocol explained in the previous section.

The protocol behind the Civitas system is briefly described here.

- **Setup Phase.** The election has a supervisor that creates and manages the election. They post the election that they created onto a public bulletin board. They additionally post public keys of the registration tellers.

  The follow it up with an *electoral roll* of the voters in the system. These *authorized* voters have a pair of keys associated with each of them, namely the *registration key* and the *designation key.*

  The tabulation tellers then jointly generate the public key for the election and post it on the bulletin board. They retain a share of the key with themselves.

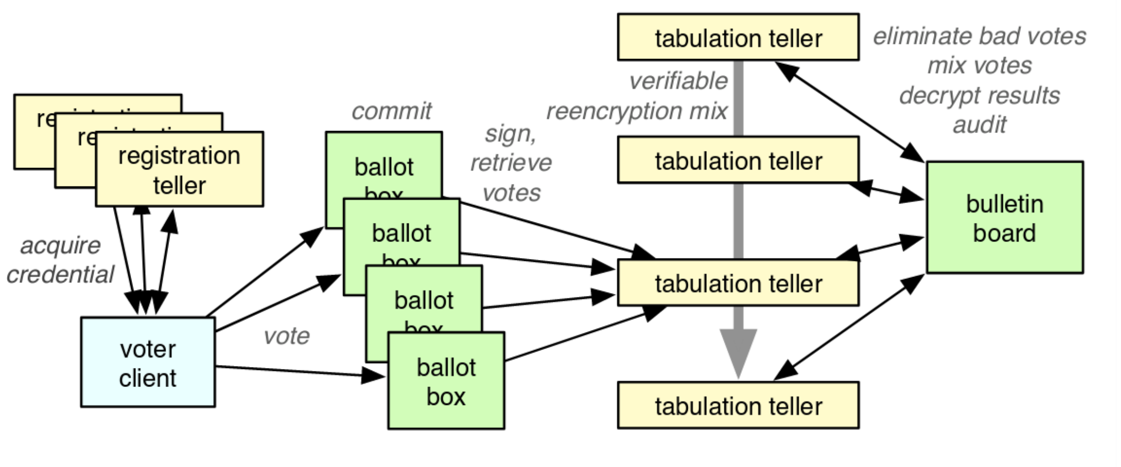


Figure 3.1: The Architecture of the Civitas [14] system is shown here. The system uses multiple registration tellers that the voters need to interact with in order to obtain their credential. They cast their vote to multiple ballot boxes so that the system does not rely on a single ballot box which would have left all votes vulnerable to loss. Multiple tabulation tellers extract and tally votes. The results are finally published along with their proofs on the public bulletin board.

- **Voting Phase.** Each voter requires a credential to participate in the election. To obtain this, they interact with multiple registration tellers who each provide a share of the overall credential. The voter and registration teller run a protocol using the voter's registration key to authenticate the voter, and the voter's designation key to provide the credential share.

  Voters resist coercion by generating a fake credential. This is done by modifying at least one of the credential shares and changing the overall proof associated with it. To aid with this, the voter needs to be assured of the honesty of at least one of the registration tellers. If this can be achieved, the system remains coercion resistant even if the coercer manages to obtain credential shares from all of the other dishonest registration tellers.

  Casting a vote to the system involves encrypting the voter's choice and their credential. They also generate two proofs of knowledge. This encrypted ballot is cast to multiple ballot boxes.

- **Tabulation Phase.** Once voting is frozen for an election, all tabulation tellers extract the votes from multiple ballot boxes and create a single list of all votes that have been cast.

They check the proofs associated with both, the credential and the choice. If either of the two proofs cannot be verified, the ballot is eliminated.

All ballot pairs are compared against each other, similar to JCJ, in an operation with a worst-case complexity of $\mathcal{O}(N^2)$. Duplicate votes are eliminated.

The remainder of the ballots need to be anonymized before verifying their credentials. They are passed through a re-encryption mixnet performed by all participating tabulation tellers. The list of valid credentials is extracted from the bulletin board and are passed through a similar random permutation.

The encrypted credential from each ballot in the list of anonymized ballots is compared against the anonymized list of valid credentials, using a Plaintext Equivalence Test (PET). Those that do not find a match are considered invalid votes and are eliminated. This operation, similar to JCJ, has a worst-case complexity of $\mathcal{O}(N^2)$.

The remaining votes are decrypted and tallied. The results, along with proofs of proper re-encryption are posted on the bulletin board.

Civitas [14] was implemented in Java and Jif. Jif [13] is a Java-based programming language that is used to control information and access flow during compile and run times. The system was developed as a backend system without a user interface.

3.5 Proposed Improvements to the JCJ Protocol

Civitas presented an example of how a coercion resistant system could be implemented and made robust with multiple ballot boxes and registration tellers. It failed to address the complexity of eliminating duplicate votes and votes with invalid credentials. The system retained the worst case scenario of $\mathcal{O}(N^2)$, similar to that of the JCJ Protocol.

Suggestions by Smith and Weber

Smith and Weber [15, 16, 17] suggested an alternative to using $\mathsf{PET}(A_i, A_j)$ for the removal of duplicate votes and $\mathsf{PET}(A_i, \sigma_j)$ for the removal of invalid votes.

All tabulation tellers can instead share a random value, z, such that, $z \leftarrow \mathbf{Z}_q^*$. Before credentials are compared, they can be raised to the power of z, and then decrypting them.

$A_1^z = \mathsf{Enc}_\epsilon(\sigma_1^z),\ \ldots,\ A_N^z = \mathsf{Enc}_\epsilon(\sigma_N^z)$

The result is that the decrypted credentials are blinded values. For example, credentaial σ_1 decrypts to σ_1^z. These values can be added to a hash table which detects collision detection. The operation will then be reduced to $\mathcal{O}(N)$.

Implementing this strategy for the removal of invalid credentials, however, introduces an attack strategy by which a coercer can identify if a vote with a specific credential was counted. Araujo et al. [19], Clarkson et al.(Civitas) [14], and Pfitzmann [57] countered this. A brief description of the attack follows.

Consider the scenario that a voter provides a credential σ to an attacker. The attacker can post two votes, one with the credential σ and one with square of the credential σ^2. When the talliers publish the blinded outputs of the mixnets, the will look for the two blinded credentials, one being the square of the other, σ^z and σ^{2z}. If they find that these two votes have been eliminated, they can identify that they have been provided with a fake credential.

Suggestions by Araujo et al.

Araujo et. al. [18, 19, 20] proposed a modification of the JCJ Protocol that was based on the group signature of Camenisch et al. [58].

Each credential in their system is composed of two parts; the short part, r, comprises around 20 ASCII characters that can be written down using a pen and paper; the long part (a, b, c), is recorded in a device or sent via email. The value, r, can be altered to furnish fake credentials.

Their scheme is designed such that duplicates can be publicly computed from the bulletin board by any entity that has access to it. Before determining illegitimate votes, the talliers mix the votes and decrypt them.

The vulnerability of this scheme comes from the fact that a majority of registration tellers can produce an illegitimate credential and publish it on the bulletin board without the voter's knowledge.

Selections

Clark and Hengartner [59] proposed Selections as a practical improvement to the JCJ protocol. The protocol was based on the use of dictionary-based passwords. The scheme was based on a previous work of theirs involving panic passwords [60].

When registering to participate in the system, a voter chooses a list of α passwords, extracted from a well-known dictionary. Thereafter, for every election, they perform a protocol with the registration tellers during which they choose an accepted password for the election. All other passwords from the dictionary are treated as panic passwords.

While this system makes it easy to handle passwords and make them readable, the system reduces the entropy associated with a credential. Given an anonymity list of 5 words from a 25000-word UNIX dictionary (as suggested by the Clark and Hengartner), the list has an entropy of 73-bits. An adversary can brute force the system to post votes on behalf of voters even if they don't realize the identity of the voter itself.

Suggestions by Spycher et al.

Spycher et al. [21] identified the vulnerabilities in the suggestion by Araujo et al. [18] and Smith et al. [15]. They made suggestions to both, improve the worst-case scenario of the JCJ Protocol and avoid introducing new vulnerabilities to the coercion resistance of the system.

Once votes with invalid proofs have been eliminated, they suggest that a random number, X_i, of votes be added for each voter in the roll. They do this because, as shown in the following paragraphs, it becomes evident that the bulletin board can reveal the number of votes associated with each voter.

For the removal of duplicate votes, they stuck to Smith et al.'s [15, 16, 17] recommendation of using a shared random value, z, by the tabulation tellers. They realized that this system does not introduce new vulnerabilities as long as the same approach is not employed for the removal of invalid votes from the system.

For the removal of invalid votes, they suggested that each vote in the *roster* (list of valid credentials) be associated with a plaintext public index, *index*. As a result, each vote becomes a three-parameter tuple, (A, B, C), where A and B are encryptions of credentials and choice (as in JCJ), and $C = \mathsf{Enc}_\epsilon(index, \alpha_C)$.

The tabulation tellers can perform a re-encryption mixnet shuffle to make every vote change place and form on the list, $(\hat{A}_i, \hat{B}_i, \hat{C}_i)$. The tellers can jointly decrypt each $\hat{C}_i$ to the plaintext index to turn vote tuples to $(\hat{A}_i, \hat{B}_i, index)$. The indices can then be replaced with the corresponding credential entry on the bulletin board, $(\hat{A}_i, \hat{B}_i, \hat{\sigma_j})$. A plaintext equivalence test can be performed on the two encrypted credentials in each vote, $\mathsf{PET}(A_i, \sigma_j)$. Any vote that fails this is eliminated. The votes that find pass the PET can be decrypted and added to the final tally.

3.6 Research Objectives

The suggestions made by Spycher et al. reduce the complexity of the JCJ Protocol down to a worst-case complexity of $\mathcal{O}(N)$, making it linear. These suggestions counter the vulnerabilities of the previous suggestions by Araujo et. al. [18, 19, 20] and Smith and Weber [15, 16, 17].

The proposal, however, missed a few caveats regarding the implementation of the system. It stuck to the JCJ proposal and did not include implementation details about the generation and transfer of credentials.

This thesis covers a coercion resistant election protocol adapts Spycher et al.'s suggestions to the JCJ Protocol and including the implementation of credential generation and transfer used by Civitas.

The proposed protocol needs to be validated to ensure that it does not introduce security vulnerabilities. It also needs to be ensured that the model complies with the definition of coercion resistance.

Algorithms are proposed for the management of the encryption and proofs for the implementation of the system. The system is then implemented as a web-based application, using the proposed protocol and the algorithms associated with it.

The system is then compared with previous protocols and systems in terms of the number of modular exponentiations and timing measurements.

Chapter 4

The Proposed Protocol

This chapter introduces Resist, a coercion resistant election protocol that derives from the suggestions made by Spycher et al. [21] to the JCJ Protocol [12]. The sections that follow provide an overview of the protocol, present the players and entities participating in the process, and present the phases covered in each election.

4.1 Overview of the Process

In this section, we cover the overall protocol from the perspective of the voter. The sections that follow help expand on the brief description provided here.

1. The voter finds their name on the Encrypted Voter List (EVL). They evaluate the possible options for an authentication teller, choose one of that they trust, and approach them with valid identity credentials.

2. They obtain their authentication and designation keys, and verify that the encryptions of these keys and their index, as posted on the bulletin board, are satisfactory. Additionally, they verify that their name, with an appropriate index has been made part of the Authenticated Voter List (AVL).

3. When an election is announced, they evaluate the possible options for registration tellers, and identify at least one honest teller. They complete a protocol with all registration tellers. They perform a handshake with the registration tellers independently and obtain multiple shares of their credential. They combine these shares to form a single, complete credential. They verify their shares with the values posted across their index on the bulletin board. Additionally, they verify that their name has been correctly included in the Registered Voter List (RVL).

4. Once the voting window has been opened, they audit as many ballots as they wish. Once they are satisfied, they post an encrypted ballot along with their credentials and index to the bulletin board.

5. The bulletin board lists all of the ballots in order of the time that they were cast. Any voter can verify that their ballot has been posted. These encrypted votes, however, do not include any identification (in plaintext) of whom the vote belongs to.

6. When the voting phase concludes, all the votes are evaluated for uniqueness and correctness. The filtered vote set is summed for the tally. Upon decryption, voters can find the results of the election, associated with appropriate proofs, on the bulletin board.

7. Any auditor (not necessarily a voter) can verify the election at any point in time thereafter by accessing data available from the bulletin board.

The voter additionally has the ability to resist coercion. Before we explain the mechanism behind this resistance, we introduce the players in the section below.

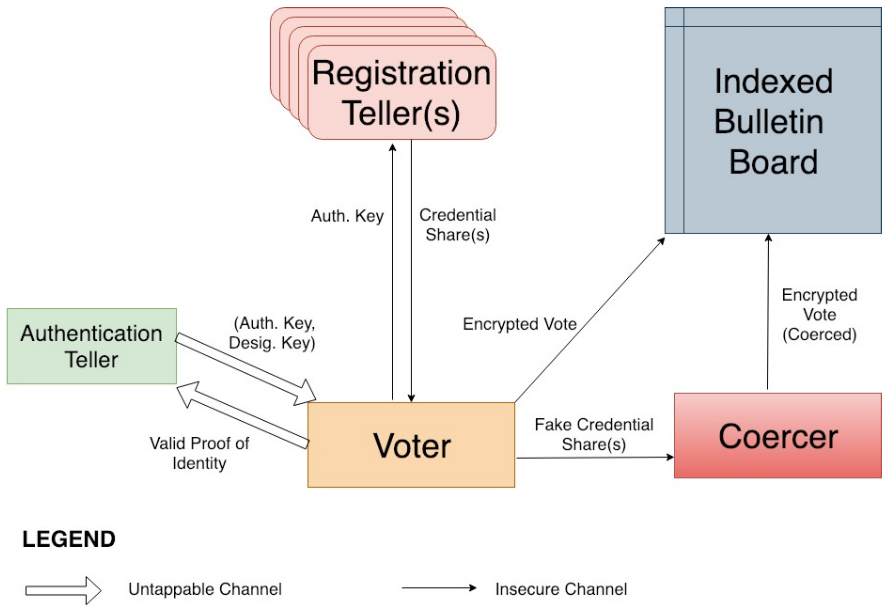


Figure 4.1: Voter Interaction with Resist's entities

4.2 Players

This section lists the different players involved in Resist's working.

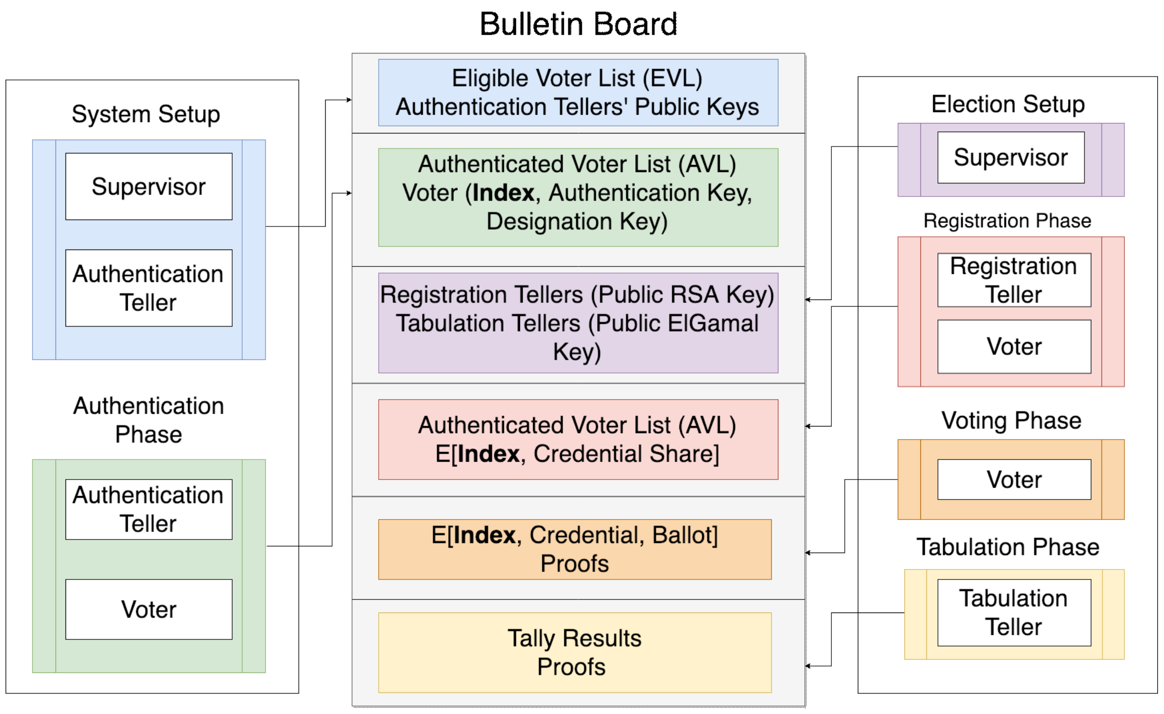

Bulletin Board
System Setup
Supervisor
Authentication Teller
Authentication Phase
Authentication Teller
Voter
Eligible Voter List (EVL)
Authentication Tellers' Public Keys
Authenticated Voter List (AVL)
Voter (**Index**, Authentication Key, Designation Key)
Registration Tellers (Public RSA Key)
Tabulation Tellers (Public ElGamal Key)
Authenticated Voter List (AVL)
E[**Index**, Credential Share]
E[**Index**, Credential, Ballot]
Proofs
Tally Results
Proofs
Election Setup
Supervisor
Registration Phase
Registration Teller
Voter
Voting Phase
Voter
Tabulation Phase
Tabulation Teller

Figure 4.2: Resist's Indexed Bulletin Board across phases

4.2.1 Supervisor

The supervisor of the election is a trusted body that creates and handles the entire system. They manage the list of voters, authentication tellers, registration tellers, and tabulation tellers. They are responsible for establishing key exchanges for each of the players.

The supervisor also conducts and manages elections. They create ballots, questions, and choices. They have the ability to start and stop elections, registration phases, voting phases, and tabulation phases. All help and queries regarding the system are directed to the supervisor.

4.2.2 Authentication Teller

The only player that solely takes part in the physical setup phase, namely, the Authentication Teller, does not necessarily have to operate during an election phase alone. Authentication tellers are usually independent of the supervisory body, so that voters who trust a single authentication teller can make a choice of approaching only them.

Authentication tellers are required to verify the identity of voters that have already been listed as eligible. Since such verification rules change with each organization and context, Resist makes no infrastructure to restriction on how such a teller needs to perform their duties.

4.2.3 Registration Teller

A registration teller is specific to an election. They operate remotely and are independent from the supervisory body governing Resist's functioning. They have the ability to verify that a voter has been authenticated to take part in the specific election in question. They generate a credential share for the voter, provide the voter with the share, and post an encrypted copy of this share on the bulletin board.

Since these bodies are untrusted and exposed to possible coercion, they are restricted in their ability to add keys to the bulletin board, in that they need to specify the index of the voter to whom the key belongs. They also need to provide a proof to the voter regarding the encrypted credential-share that they have posted on the bulletin board.

At least one registration teller is assumed to be honest and their honesty known to the voter. In the event that all other tellers collude with the coercer, the voter can alter the credential share of the honest teller alone.

4.2.4 Voter

The voter is the primary entity in an election. They obtain keys for multiple elections from the authentication teller of their choice. They obtain election-specific credentials from registration tellers. They take part in the election with the intention to cast valid votes.

In the event of coercion, they generate fake credentials based on the credential shares in their possession. They pass on this fake credential to the coercer, and continue with casting a valid vote as they had initially intended.

4.2.5 Coercer

This is the primary form of an adversary that Resist intends to tackle. They may coerce a voter physically in a remote election system. They may make a voter do one of the following tasks.

1. Abstain from voting (either by oversight or by obtaining their credentials)
2. Cast a specific ballot (of the coercer's choice)
3. Vote on behalf of the voter

Additionally, the coercer may attempt to influence registration tellers and tabulation tellers. Resist counters this by increasing verifiability by the need to publish proofs. In the event that a proof is incorrect, voters can identify coerced tellers.

4.3 Setup Phases

4.3.1 System Setup Phase

The initial setup of the voting system assumes the trustworthiness of the election supervisor. Prior to beginning the actual election process, a system like Resist involves cementing the inclusion of a few key attributes and players of the system. This phase requires an untappable channel which should imply the physical presence of key players.

Supervisor Setup

The supervisor operates as the administrator for all elections held within the system. The supervisor additionally set up authentication tellers individually. The key exchanges associated in this assignment of authority requires untappable channels. Any independent entity should be capable of serving as an authentication teller, given a prerequisite satisfaction of trustworthiness. This requisite can be evaluated subject to the guidelines laid down by the supervisory authority of concern.

As a governing body, the supervisor posts a list of eligible voters obtained from a precompiled database. A real-world analogy for this will be a subset of a reliable census, including only people of the right age and citizenship. Within Resist, this list is called the ”Eligible Voter List” (EVL). The EVL serves as a superset of all other voter lists within the system.

Voter Authentication

Once a voter has been enrolled in the EVL, they then approach the authentication teller of their preference. On provision of valid identity and after appropriate verification, the authentication tellers can then provide them with two keys, one termed as the authentication key and the other termed as the designation key. The public keys are made available on the bulletin board. The voter is then entered into an ”Authenticated Voter List.” These two keys have their corresponding roles and counterparts in the Civitas protocol. Additionally, voters also receive a uniquely identifiable index on the bulletin board. This is intended to eventually reduce the tabulation time for each election.

Once a voter has obtained their keys, they can use these keys to participate in multiple elections thereafter. These keys are modifiable and updatable in the database on the request of the voter. Dissociating this phase from the election phase makes the interactions from this phase available at all times, regardless of election periods.

4.3.2 Election Phase

Once the system has been set up, voters are available to participate in elections. The supervisor can then create and conduct elections, with guaranteed participation. Conducting an election requires two other key players, the registration tellers and the tabulation tellers. This phase can be further divided into the following phases in order.

Election Creation Phase

In this phase, the supervisor sets up election parameters and players. They have to provide the election with a name, a list of questions (with fixed choices), identities of registration tellers, and identities of tabulation tellers.

The phase includes a key exchange with each registration teller and tabulation teller, at the end of which, their public keys are posted to the bulletin board. This key exchange can preferably be conducted in-person, but is still considered a remote phase, since the voter does not need to be physically present.

Both registration tellers and tabulation tellers are independent entities. Each registration teller receives a unique key that gives them the authority to create and post credential shares. Tabulation tellers, however, receive a share of the overall tabulation key. This is implemented with the intention of using a threshold cryptosystem, wherein the decryption of votes requires the minimal agreement of a pre-decided key-share.

Registration Phase

Voters have an authentication key and a designation key at this point. These two shares give them access to their private credential for each election. They get to choose the registration tellers that they trust and intend to obtain their keys from. Since a single voter may not completely trust an independent authority, they can obtain shares of their credential from multiple tellers.

Each voter sends their authentication key to each of the registration tellers of their choice. After a simple verification of the key, the tellers generate and return their key, which can only be decrypted with the voter's designation key. The registration tellers also post voter credential shares to the bulletin board, encrypted with the tabulation tellers' public key.

Registration tellers additionally add registered voters to the "Registered Voter List" (RVL) which serves as a subset of the previously posted AVL. This adds to Resist's verifiability.

4.4 Indexed Bulletin Board

Resist's central verifiability is derived from the availability of a publicly accessible bulletin board. The board is used in each phase of the system, and besides verifiability, is the only medium of passing information between two distinct phases.

1. **System Setup Phase.** All voters that are eligible to participate in the election are posted to the bulletin board as part of the EVL. The supervisor also posts public keys of the authentication tellers.

2. **Authentication Phase.** Each authentication teller verifies voters that approach them with valid verification (or identity) credentials. The authentication teller posts the public part of the voter's authentication and designation keys onto the bulletin board, associated with a unique index for each voter. Voters are also added to the AVL on the bulletin board.

3. **Registration Phase.** Each share of each voter's credential is posted on the bulletin board, encrypted with the public key of the tabulation tellers. These credential shares are associated with a voter's index. Additionally, all voters that receive a credential are added to the RVL.

4. **Voting Phase.** All ballots cast during an election are posted in an encrypted format, along with associated proofs. The encrypted ballots are available on the bulletin board in order of the time that they were cast.

5. **Tabulation Phase.** Once the ballots have been filtered and tabulated, the tabulation tellers post the final results of the election on the bulletin board, along with requisite proofs.

Once a value has been posted on the bulletin board, it cannot be altered or modified. The only modifiable entities are the three voter lists, that need to be updated as time passes.

4.5 Ballot Casting

A voter can access a verifiable copy of the entire ballot and mark their individual choices for each question. Once the ballot has been completely marked, it is encrypted with the public key of the tabulation tellers. The encrypted ballot includes individual proofs generated for both, the ballot and the index/credential.

Resisting Coercion. In the event that an attempt is made to coerce the vote, the voter can simple generate fake credentials and provide the same to the coercer. Resist provides the ability to generate an associated proof that makes their credential indistinguishable from the rest. The coercer may then cast a vote on the voter's behalf, using the fake credential.

Revoting. It is assumed that the voter has enough time to cast at least a single ballot without the knowledge of the coercer. Additionally, the voter can cast multiple ballots associated with the same credential. This especially helps if the coercer manages to obtain valid credentials for the voter, and the honest voter wishes to override such a vote. The rules for tallying duplicate ballots can be explicitly specified by the supervisor at the time of creation of the election.

4.6 Tallying

The tabulation authorities initially need to eliminate incorrectly formed votes according to the rules decided at the start of the election. A single run through the votes should suffice, given that each ballot explicitly provides zero knowledge proofs for each vote.

The tabulation tellers then add a random number of votes against each voter on the roll. They then pass the votes through a mixnet to anonymize them.

This credentials from this anonymized list are then blinded and entered into a hash table. Collisions are silently dropped and the result is a list of unique votes, with duplicates eliminated.

The unique votes are then passed through a second mixnet. The indices from the remaining list are decrypted and replaced with corresponding credentials from the voter roll. Any vote found to have credentials different from those on the roll is eliminated.

The remaining votes, which are now unique, honest, and valid, are decrypted. Using the combined decryption key from multiple tabulation tellers, the final tally for each of the choices is revealed.

The results of the election are posted on the bulletin board, along with their associated proofs. Anyone who can access the bulletin board can verify that the results have been tallied and the proofs have been generated correctly.

Chapter 5

Verifying the Proposed Protocol

5.1 Overview of ProVerif and Applied Pi Calculus

Security protocol evaluation is used to formally verify models for claimed properties. Automatic verification of security protocols are performed using such as Scyther [61], ProVerif [62], and AVISPA [63] among others. Each of these tools satisfy a particular type of security protocol, depending on their implementation and definition.

Since Proverif [62] checks for observational equivalence, it is used to evaluate the (in)ability of an adversary (the coercer) to differentiate between two processes, which are explained later in this section. However, Proverif checks for a stronger notion of equivalence than is fulfilled by ballot secrecy [64].

Formal definitions of electronic voting protocols, using Applied Pi calculus, can be found in the works of Deluane et al. [65, 66, 67]. This eventually lead to a more specific verification of the JCJ Protocol [12] in the work of Backes et al. [1]. The analysis performed in their work verifies their definitions voter privacy, receipt-freeness, and coercion resistance. Relevant examples of verification of other internet voting protocols that were carried out using applied pi calculus include contributions from Gerling et al. [68] for Civitas [14] (developed on Backes et al. [1]), Acquisti [69] by Meng et al. [70], and the Norwegian Internet Voting Protocol [64] by Cortier et al. [71].

Resist's voting protocol is based on Spycher et al.'s [21] suggested improvement of the JCJ/Civitas protocol [12]. Since Backes et al. [1] analyzed and validated the JCJ protocol, Resist's verification involved building upon this work to include an indexed bulletin board, and its implication on soundness, receipt freeness, and coercion resistance.

Resist's model for Verification can be seen to have the use of indices associated with voter credentials, leading to changes primarily in zero knowledge proofs.

5.2 Syntax of Applied Pi Calculus

This section provides an overview of the semantics of the applied pi-calculus used in definitions below.Please refer Section 2 and Section 5.1 of Backes et al. [1] to gain a clear understanding of the notations. A signature $\sum$, defines a finite se of function symbols, each with an arity. The function with arity 0 represents a constant symbol. Given $\sum$, an infinite set of terms an be built as follows. These terms present naming conventions and are not used to define variable names.

L, M, N, T, U, V ::=	terms
a, b, c	channel names
n, m	names of any sort
x, y, z	variable
u	names and variables
$f(M_1, ..., M_t)$	function application

The terms above, are equipped with the basic equational theory, E, of equivalence relations, closed under substitution relations on terms as variables and under application of term contexts. E is written for equality as $E \vdash M = N$ and for inequality as $E \nvdash M = N$. The equational theory given below contains function symbols taken from previous work by Backes et al.[72].

$\mathsf{eq}(x, x) = \mathsf{true}$

$\wedge\ (\mathsf{true},\ \mathsf{true}) = \mathsf{true}$

$\vee\ (\mathsf{true}, x) = \mathsf{true}$

$\vee\ (x, \mathsf{true}) = \mathsf{true}$

$\mathsf{pet}(\mathsf{enc}(x, \mathsf{pk}(y), z), \mathsf{enc}(x, \mathsf{pk}(y), w), \mathsf{sk}(y)) = \mathsf{true}$

$\mathsf{fst}(\mathsf{pair}(x, y)) = x$

$\mathsf{snd}(\mathsf{pair}(x, y)) = y$

$\mathsf{dec}(\mathsf{enc}(x, \mathsf{pk}(y), z), \mathsf{sk}(y)) = x$

$\mathsf{msg}(\mathsf{sign}(x, \mathsf{sk}(y))) = x$

$\mathsf{ver}(\mathsf{sign}(x, \mathsf{sk}(y)), x, \mathsf{pk}(y)) = \mathsf{true}$

The syntax for an example process, P, is as below.

- Plain processes are represented with specifications as $vn.P$, which generates a fresh name n before behaving as process P.
- If a message N is to be receives by P on channel a, it is represented as $a(x).P$, which makes the process behave as $P\{N/x\}$ (N given x).
- If a message N is output on channel a before process P, this is represented as $\bar{a}\langle N\rangle.P$.
- $!P$ represents an unbounded number of copies of P.
- $P|Q$ executes P and Q parallely.

The scopes of names and variables is elaborated below.

- $fn(P)$ represents free names in process P.
- $fv(P)$ represents free variables in process P.
- $\tilde{M}$ represents an arbitrary sequence of terms $M_1, M_2, ..., M_k$.
- $v\tilde{n}$ is an arbitrary sequence of name restrictions $n_1, n_2, ..., n_k$.
- $\bar{c}\langle\tilde{x}\rangle.P$ is the output of a tuple $\tilde{x}$ on channel c before behaving as process P.

Consider the following readable example, $let\ x = M\ in\ P$. It is formally represented as $va.(\bar{a}\langle M\rangle|a(x).P)$. Similarly, a statement, $let\ x \in \tilde{M}inP$, is formally represented as $va.(\bar{a}\langle M_1\rangle|\bar{a}\langle M_2\rangle|...|\bar{a}\langle M_k\rangle|a(x).P)$, where $a \notin fn(P)$.

The definitions below are modelled using the syntax of processes, variables, names, and channels, as before, in terms of execution traces of events. We point the interested reader to Abadi et. al. [73], to better understand the concept of execution traces.

5.3 Modeling and Verifying the Protocol

5.3.1 Soundness

Backes et al. described soundness with the help of correspondence assertions [74], an approach that imposes causality between the occurrence of events in the election protocol. The soundness of a protocol is adjudged on the basis of three properties, which are defined below.

The events in context include newid(in) when an identity is issued for a voter, startid(id) and startcorid(id) are the starts of registration for an honest and a corrupted voter (one that colludes with the coercer).

The event beginvote(id, v) is the casting of a vote v by an honest voter with identity id. A corrupted voter, in this model, casts a vote without asserting a choice v. The event endvote(v) represents the tallying of the vote v.

Definition 5.3.1. Soundness. A trace t guarantees soundness if and only if the following conditions hold:

1. **Inalterability.** The inability to change the contents of a vote. This is guaranteed by ensuring that every vote that is included in the final tally corresponds to a vote cast by an honest voter and a corrupted voter.

   For any t_1, t_2, v such that $t = t_1 :: \mathsf{endvote}(v) :: t_2$, there exist id, t', t'', t''' such that

   (a) an honest voter
   $t_1 = t' :: \text{startid}(id) :: t'' :: \text{beginvote}(in, id) :: t'''$ and $t' :: t'' :: t''' :: t_2$ guarantees soundness

   (b) or a corrupted voter
   $t_1 = t' :: \text{startcorid}(id) :: t''$, and $t' :: t'' :: t_2$ guarantees soundness

2. **Eligibility.** Only registered voters can cast a ballot. Implied by correspondence assertions between registration and casting, shaping the process to enforce causality.
   For any t_1, t_2, id, such that, $t = t_1 :: \text{startid}(id) :: t_2$, or $t = t_1 ::$ startcorid(id) $:: t_2$, the event newid(id) occurs in t_1.

3. **Non-reusability.** Ensuring that a voter is responsible for a single valid vote only. An election protocol can guarantee non-reusability by ensuring an injective relationship between the start and end of casting a ballot.
   For any t_1, t_2, id, such that, $t = t_1 :: \text{startcorid}(id) :: t_2$, the events startid($id$) and startcorid($id$) do not occur in $t_1 :: t_2$.

Verifying the formal definition of soundness required annotation similar to Table 5.1. The inclusion of an index, implied additional message exchanges, and primarily, a modification to the existing parameters of previously defined zero knowledge proofs. The results obtained from ProVerif show that Resist's protocol guarantees inalterability, eligibility, and non-reusability despite mild alterations made to existing annotations.
Although Resist, in its implementation, guarantees non-reusability in the registration phase using Civitas' [14] implementation of Needham-Schroeder-Lowe [75], this was altered here by the modelling of a simple nonce handshake, which satisfies the purpose of analyzing the protocol as a whole. An analysis of Needham-Schroeder-Lowe can be found in the works of Bogdan Warinschi [76].

5.3.2 Coercion Resistance

Resist's definition of coercion resistance is derived from the initial proposal by Juels et al. [12] in their description of the JCJ Protocol. We have described them below to provide an overview before formal analysis.

1. **Receipt-freeness.** Since the Australian ballots' passive privacy did not guarantee resistance to coercion, receipt freeness, as a concept, was first introduced by Benaloh and Tunistra [77]. This property guarantees the inability of a voter to prove how they voted.

2. **Immunity to simulation attacks.** This attack definition drives the formal analysis and verification since it implies both, receipt-freeness and forced abstention. A protocol that guarantees resistance to coercion leaves the coercer incapable of differentiating whether or not they have been coerced.

3. **Immunity to forced-abstention attacks.** A forced abstention attack is one wherein a voter is forced to abstain from casting their ballot in an election. This attack is almost impossible to solve in physical elections, but can potentially be solved using remote voting.

4. **Immunity to Randomization attacks.** A randomization attack is often witnessed when a voter is forced to enter a random message as part of their

Table 5.1: Resist (or Indexed Juels, Catalano, and Jakobsson) Processes in the Applied Pi-calculus, derived from Backes et al. [1]

voter $\triangleq$
$c_{id}(id)$
startid(id)
chVR$(index, cred)$
let $vote \in v_A, v_B, v_C$ in
beginvote$(id, vote)$
$vr_1, vr_2, vr_3.\overline{pub}\langle zk\rangle$

corvoter $\triangleq$
$c_{id}(id)$.
startcorid(id).
chVR$(index, cred)$.
$\overline{pub}\langle index, cred\rangle$

identityissuer $\triangleq$
v_{id}.
$new_{id}(id)$.
$\overline{c_{id}}\langle id\rangle$.
$\overline{chIR}\langle id\rangle$.
$\overline{pub}\langle id\rangle$

registrar $\triangleq$
$chIR(id)$.
$vcred.\ vindex.\ vr_1.\ vr_2$.
$\overline{chVR}\langle index, cred\rangle$.
$\overline{pub}\langle sign(enc(index, pk(kT), r_1), sk(kR)), sign(enc(cred, pk(kT), r_2), sk(kR))\rangle\rangle$.
$\overline{chRT}\langle enc(index, pk(kT), r_1), enc(cred, pk(kT), r_2))\rangle$.

tallier $\triangleq$
$pub(zkp)$.
$if\ Ver_{6,7}(proof, zkp)\ then$
$if\ Public_5(proof) = v_A\ then$
$if\ Public_6(proof) = v_B\ then$
$if\ Public_7(proof) = v_C\ then$
$let\ encindex = Public_1(zkp)\ in$
$let\ enccred = Public_2(zkp)\ in$
$let\ encvote = Public_3(zkp)\ in$
$chRT((encindex_1, enccred_1))$
$if\ \mathsf{eq}(\mathsf{dec}(encindex, sk(kT)), \mathsf{dec}(encindex_1, sk(kT)))\ then$
$if\ \mathsf{pet}(enccred, enccred_1, sk(kT))\ then$
$let\ vote = \mathsf{dec}(encvote,\ sk(kT))\ in$
$\overline{c_{votes}}\langle vote\rangle$

JCJ $\triangleq$ $vc_{id}.vchIR.vchVR.vchRT.vkT.vkR.\overline{pub}\langle\ \text{pair}(pk(kT), pk(kR))\rangle$.
(!voter | !corvoter | !identityissuer | !registrar | !tallier)

$zk =$ $\text{ZK}_{6,7}(index, r3, cred, r2, vote, r1;$
$\mathsf{enc}(index, pk(kT), r_2), \mathsf{enc}(cred, pk(kT), r_3), \mathsf{enc}(vote, \text{pk}(kT), r_1), pk(kT), v_a, v_b, v_c;$
$proof)$

$proof =$ $\text{enc}(\alpha_1,\ \beta_4,\ \alpha_2) = \beta_3 \wedge$
$\text{enc}(\alpha_3,\ \beta_4,\ \alpha_4) = \beta_2 \wedge$
$\text{enc}(\alpha_5, \beta_4, \alpha_6) = \beta_1 \wedge$
$(\alpha_5 = \beta_5 \vee \alpha_5 = \beta_6 \vee \alpha_5 = \beta_7)$

LEGEND :
chVR - Private channel between *voter* and *registrar*;
chIR - Private channel between *identityissuer* and *registrar* ;
chRT - Private channel between *registrar* and *tallier*;

vote. Elections that provide the option for write-ins are exploited to cast votes for random candidates that are not on the candidate list. Resist does not need to address this since, in its implementation, it does not provide the option for

Table 5.2: Resist (or Indexed Juels, Catalano, and Jakobsson) Processes for Coercion Resistance, derived from Backes et al. [1]

coercedvoter $\triangleq$
$c_{id}(id).$
$chVR(index, cred).$
$\overline{c}\langle(index, cred)\rangle.$
$\overline{c_1}\langle(index, cred)\rangle.$

cheatingvoter $\triangleq$
$c_{id}(id).$
$chVR(index, cred).$
$vfakeindex.$
$vfakecred.$
$\overline{c}\langle(fakeindex, fakecred)\rangle|$
$let vote = v_A in$
$vr_1.vr_2.vr_3.$
$\overline{pub}\langle zk\rangle$

absvoter $\triangleq$
$c_{id}(id).$
$chVR(index, cred).$

tallierE $\triangleq$
$pub(zkp).$
$if\ Ver_{6,7}(proofenc,\ zkp)\ then$
$if\ Public_5(proof) = v_A\ then$
$if\ Public_6(proof) = v_B\ then$
$if\ Public_7(proof) = v_V\ then$
$let\ encindex = Public_1(zkp)\ in$
$let\ enccred = Public_2(zkp)\ in$
$let\ encvote = Public_3(zkp)\ in$
$chRT(encindex_1, enccred_1).$
$\overline{c_2}\langle(encindex, encindex1, enccred, enccred1, encvote)\rangle.$
$if\ \mathsf{eq}(\mathsf{dec}(encindex, sk(kT)), \mathsf{dec}(encindex_1, sk(kT)))\ then$
$if\ \mathsf{pet}(enccred, enccred_1, sk(kT))\ then$
$letvote = \mathsf{dec}(encvote, sk(kT))\ in$
$\overline{c_{votes}}\langle vote\rangle.$

extractor $\triangleq$
$(c_{id}(id).chVR(indexext, credext).$
$vA.vB.$
$(c_1(fakeindex, fakecred).!\overline{a}\langle(fakeindex, fakecred)\rangle)|$
$(!c_2((encindex, encindex_1, enccred, enccred_1, encvote)).$
$a(fakeindex, fakecred).$
$let\ vote = \mathrm{dec}(encvote, sk(kT))\ in$
$let\ index = \mathrm{dec}(encindex, sk(kT))\ in$
$let\ index_1 = \mathrm{dec}(encindex_1, sk(kT))\ in$
$let\ cred = \mathrm{dec}(enccred, sk(kT))\ in$
$let\ cred_1 = \mathsf{dec}(enccred_1, sk(kT))\ in$
$if\ index = fakeindex\ then$
$if\ index_1 = indexext\ then$
$if\ cred = fakecred\ then$
$if\ cred_1 = credext\ then$
$\overline{b}\langle vote\rangle)|$
$(b(z).if\ z \in \{v_a, v_b, v_c\}\ then$
$[\ ]))$% either 0 or $\overline{c_{votes}}\langle z\rangle$

JCJ–CR1 $\triangleq$ $vc_1.vc_2.vc_{id}.vchIR.vchVR.vchRT.vkT.vkR.$
$\overline{pub}\langle \mathrm{pair}(\mathrm{pk}(kT), \mathrm{pk}(kR))\rangle.$
(!voter | !corvoter | !identityissuer | !registrar | !tallierE |
coercedvoter | voter(v_A) | extractor(0))

JCJ–CR2 $\triangleq$ $vc_1.vc_2.vc_{id}.vchIR.vchVR.vchRT.vkT.vkR.$
$\overline{pub}\langle \mathrm{pair}(\mathrm{pk}(kT), \mathrm{pk}(kR))\rangle.$
(!voter | !corvoter | !identityissuer | !registrar | !tallierE |
cheatingvoter | voterabs | extractor($c_{votes}\langle z\rangle$))

write-ins.

Resist's verification for coercion mimics the analysis described in Backes et al [1]. The formal definition for coercion resistance describes two possible contexts of the protocol.

- One context represents a scenario in which the coercer successfully obtains a valid credential from the voter and either votes on their behalf or abstains from voting.
- Alternatively, the second context represents a scenario in which the voter generates and provides a fake credential to the coercer and votes on their behalf

(or voluntarily abstains from voting). The protocol can claim to be coercion resistant if the coercer fails to differentiate between these two traces.

Modelling an election protocol for coercion resistance also requires the construction of an additional context called 'extractor' (E). Once the tabulation authorities eliminate duplicate votes, the extractor identifies coerced votes (those with incorrect credentials) and balances these votes in the final tally in such a way that the coercer has no way of identifying this event. In order to identify coerced votes, the extractor needs to have access to election secrets and is thus restricted both, syntactically and semantically [1].
The verification of Resist's voting protocol, similar to JCJ, required modification of processes referred to in Table 1 to include the extractor and cheating, coerced, and abstained voters. The processes can be found in Table 2. The authors claim no novelty of this representation, since it is a version of the model presented in Backes et al. [1]. Besides, Definition 3.2 is a re-iteration of the definition from Backes et. al., tied with an overview and results from the execution of the modified version to suit Resist's protocol. The zero knowledge proofs were compiled using an earlier work of Backes et al. [72].

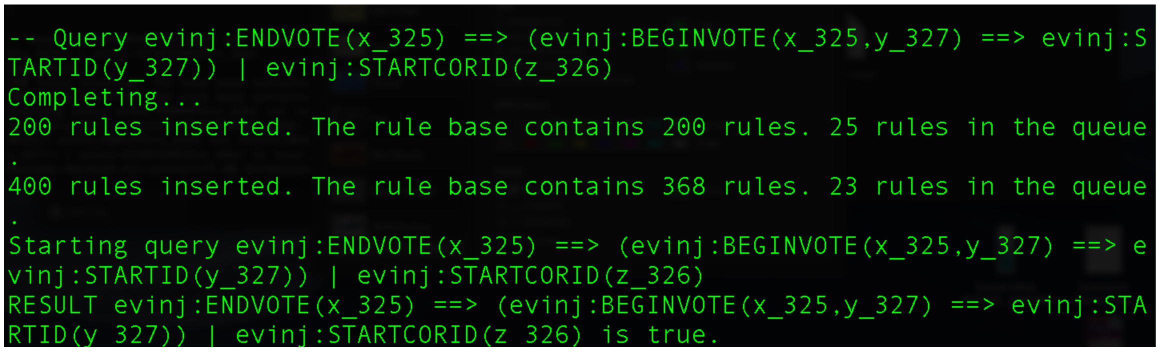
Definition 5.3.2. Coercion Resistance. [1] An election context S guarantees coercion-resistance if there exist channels c, $c1$, and $c2$, a sequential process V_{fake}, an extractor $E_k^{c1,c2,z}$, and an election context S', such that

1. There exists an election context S'' and two authority processes A, A' such that $S \equiv S''[A \mid [\,]]$, $S' \equiv vc_1,\ c2.S''[A'|[\,]]$, and $vc2.(A' \mid !c_2(x)) \approx A$;

2. $S'[\ V_i^{coerced(c,c1)} \mid V_j(v') \mid E_k^{c1,c2,z}[0]\ ]$
$\approx S'[V_i^{cheat(c,c1)}(v') \mid V_j^{abs} \mid E_k^{c1,c2,z}\ [\bar{c}_{votes}\langle z\rangle]\ ]$
where $v' \in \widetilde{v}$ is a valid vote;

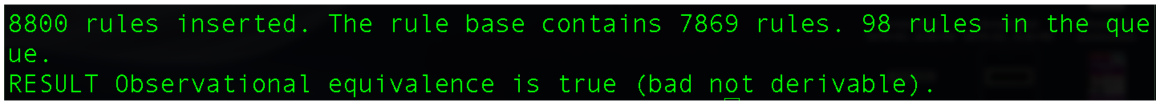
3. $vc.S'[!c(x) \mid V_i^{cheat(c,c1)}(v') \mid V_j^{abs} \mid E_k^{c1,c2,z}\ [\overline{c_{votes}}\langle z\rangle]] \approx S[V_i(v') \mid V_j^{abs} \mid V_k^{abs}]$;

4. *Let* $P = c(\widetilde{x}).let\ x_v = v\ in\ V^{vote}\ \widetilde{x}/\widetilde{u},\ v \in \widetilde{v},\ \widetilde{u} = captured(V^{reg}),$ *and* $\widetilde{x} \cap \widetilde{u} = \emptyset$ *then*



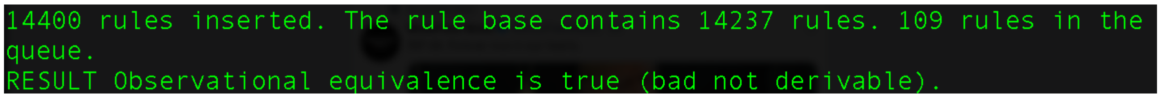
```
-- Query evinj:ENDVOTE(x_325) ==> (evinj:BEGINVOTE(x_325,y_327) ==> evinj:S
TARTID(y_327)) | evinj:STARTCORID(z_326)
Completing...
200 rules inserted. The rule base contains 200 rules. 25 rules in the queue
.
400 rules inserted. The rule base contains 368 rules. 23 rules in the queue
.
Starting query evinj:ENDVOTE(x_325) ==> (evinj:BEGINVOTE(x_325,y_327) ==> e
vinj:STARTID(y_327)) | evinj:STARTCORID(z_326)
RESULT evinj:ENDVOTE(x_325) ==> (evinj:BEGINVOTE(x_325,y_327) ==> evinj:STA
RTID(y_327)) | evinj:STARTCORID(z_326) is true.
```

(a) ProVerif output for Soundness



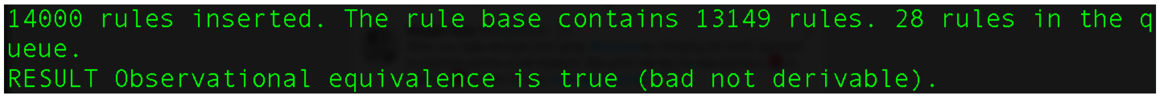
```
8800 rules inserted. The rule base contains 7869 rules. 98 rules in the que
ue.
RESULT Observational equivalence is true (bad not derivable).
```

(b) ProVerif output for Condition 2 of Coercion Resistance



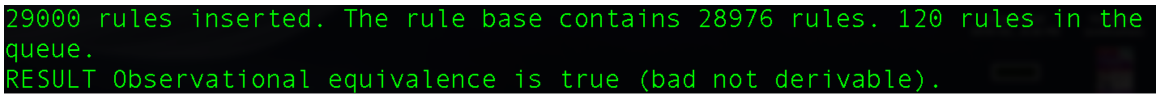
```
14400 rules inserted. The rule base contains 14237 rules. 109 rules in the
queue.
RESULT Observational equivalence is true (bad not derivable).
```

(c) ProVerif output for Condition 3 of Coercion Resistance



```
14000 rules inserted. The rule base contains 13149 rules. 28 rules in the q
ueue.
RESULT Observational equivalence is true (bad not derivable).
```

(d) ProVerif output for Condition 4 of Coercion Resistance



```
29000 rules inserted. The rule base contains 28976 rules. 120 rules in the
queue.
RESULT Observational equivalence is true (bad not derivable).
```

(e) ProVerif output for Condition 5 of Coercion Resistance

Figure 5.1: Validating Resist for Soundness and Coercion Resistance using ProVerif

$$\begin{aligned}&\nu c.S'[P \mid V_i^{cheat(c,c1)}(v') \mid V_j^{abs} \mid E_k^{c1,c2,z}[\overline{c_{votes}}\langle z\rangle]] \approx\\&\nu c.S'[P \mid V_i^{cheat(c1,c2)}(v') \mid\\&V_j^{abs} \mid E_k^{c1,c2,z}[\overline{c_{votes}}\langle v\rangle]];\end{aligned}$$

5. $S[V_i^{inv-reg}] \approx \nu c_{votes}.(!c_{votes}(x) \mid S[V_i(v)])$, where v is a valid vote.

Condition 1 defines two election contexts S and S'; the two contexts differ only in the ability of the tallying (tabulation) authority to share a channel with the extractor.

In Condition 2, the former process represents a coerced voter (one that provides the coercer with the actual credential), corresponding to another voter casting their vote (v') and the extractor nullifying the vote. The latter process is equivalent to a voter providing a fake credential to the coercer before casting a valid vote (v'), in parallel with another voter that abstains from voting, and an extractor tallying the invalid vote that the coercer cast on behalf of the voter. In order to be coercion resistant, the two processes need to be observationally equivalent. This condition was verified with ProVerif.

While Condition 2 is a basic representation of observational equivalences, the conditions that follow include additional representations of the characteristics of the extractor.

Condition 3 represents a coercer (with a fake credential) that abstains from voting. As a result, the extractor, which has access to the fake credential, must abstain. This condition was verified with ProVerif.

Condition 4 represents a coercer using fake credentials to cast a valid vote on behalf of the voter, Vi, followed by the extractor tallying this vote. Resist was verified to satisfy this equivalence.

The final condition includes an additional restriction to justify the extractor's abstraction of the third voter; the tallying authority silently discards any vote with invalid registration credentials. This was verified using ProVerif.

Chapter 6

Algorithms in the Proposed Protocol

The chapters so far have described Resist's protocol and modeled it using observational equivalences. The phases and interaction between the tellers, however, need to be broken down into protocols and algorithms before going ahead with the implementation of the system (described in Chapter 7).

This chapter presents the primitives of cryptography and describes the algorithms and protocols used for encryption, decryption, proofs, and authentication. It presents the system from an algorithmic standpoint, which will not only help the reader understand the functioning of the protocol, but also translate this understanding into an implementation (in any language) of their preference.

6.1 Concepts in Cryptographic Systems

6.1.1 Public Key Cryptography

Initially suggested as a key exchange protocol in their paper by Whitfield Diffie and Martin Hellman [78] in 1976, public key cryptography was introduced as an alternative to symmetric key cryptography by Rivest, Shamir, and Adleman introduced RSA in 1978 [79].

The concept of public key cryptography deviated from traditional secret key cryptography in that it didn't rely on sharing a secret between two untrusted parties. It instead suggested that each entity have two keys that belong to them, one that they keep secret (and use for decryption) and one that they share with other entities (to be used for decryption). Public key cryptography found use not just in cryptography, but also in signing messages, authenticating participants in a system, and verifying the integrity of an encrypted message.

The algorithm explained in this thesis concentrated on the ElGamal public key cryptosystem, since it has been used in the implementation of the election systems.

Reasons for this choice can be found in the sections that follow. It may suffice to state that the system was developed based on the discrete logarithm problem, which also formed the basis of RSA [80].

The Discrete Logarithm Problem

Before we explain the discrete logarithm problem, it'll help to give a brief of the preliminaries. Most operations in Public Key Systems involve numbers that fall under a multiplicative cyclic group (referred henceforth as G). Every element, h in G can be generated represented as g^x for some value x. The value g is referred to as the 'generator' of G, because it generates every value in the group.

The order of the cyclic group is the number of elements in the group $\langle g \rangle$. Therefore, the elements in the cyclic group G of order n are $G = \{e, g, g^2, ..., g^{n-1}\}$, e is the identity element.

Given a group G, generator g, element $h \ni h = g^x \texttt{ mod } n)$, the value x is called the discrete logarithm of h to the base g. The discrete logarithm problem is to find the discrete logarithm (x) of an element in G, given the order of the cyclic group (n), the generator(g), and the element itself (h). For large prime numbers (size of n), there is no known efficient algorithm that solves the problem for a value of h.

Parameter Generation

Every entity in a public key cryptosystem requires a public key and a corresponding private key. The parameters are generated (given a previously chosen bit-length) prior to key generation. These parameters define the aforementioned cyclic groups, under which all values required for the cryptosystem exist.

The key parameters in an ElGamal cryptosystem involves a multiplicative cyclic group comprising integers modulo a prime value $p \ni p = 2kq+1$, where q is a randomly chosen prime. All plaintext integer messages that require encryption belong to the message space $\mathcal{M}$, which is the order q subgroup of $\mathbf{Z}_p^*$. The algorithm to generate parameters required for an ElGamal cryptosystem has been described in Algorithm 6.1.1.

The bit-length of an integer, although not represented its binary form, is referred to in bits. This is because the number of random bits in the parameters determine the entropy of the key. Usually, an ElGamal or an RSA system will have parameters of bit length 256, 512, 1024, 2048, or 4096.

Once the parameters, p, q, and g have been generated, they are used to in all encryption, hashing, and proof algorithms that are needed by the system.

Algorithm 6.1.1: ElGamal Parameter Generation

Due to : Adaptation of DSA parameter generation as stated in NIST[81].

Input :

1. L : The requested length for p (in bits), recommended $L = 2048$.
2. N : The requested length for q (in bits), for $L = 2048$, $N = 224$.

Output:

1. p, q : The requested primes, such that $|p| = L$ and $|q| = N$.
2. g : The generator of the finite field.

Process:

```
1. Select a random $N$-bit prime $q$.
2. Select a random $L$-bit number $p$. Round $p - -1$ down to a multiple of $2q$ by
   setting $p$ equal to $p - -(p \bmod q) + 1$. Unless $p$ is prime $|p| = L$. Repeat for
   $2^{\log_2(l)+2}$ tries, after which, pick a new $q$.
3. repeat
       $h \leftarrow [1...p - 1]$; $g = h^{(p-1)/q} \bmod p$;
   until $g \not\equiv \pm 1 (mod\ p)$;
4. Output($p$, $q$, $g$)
```

Key Generation

Each participant (Sender or Receiver) in a Public Key Cryptosystem has at least one key pair, one public key and one private key. The public key is used revealed to all possible senders so that they can encrypt their message using the public key. The private key is kept secret to allow only the entity, the receiver that it was intended for, to decrypt and read the message.

The key generation algorithm, Algorithm 6.1.2, shows how the discrete logarithm problem is leveraged to generate ElGamal Keys.

Algorithm 6.1.2: ElGamal Key Generation

```
Due to : Taher ElGamal [82].
Input :
    1. $p$, $q$, $g$ : ElGamal Parameters generated in Algorithm 6.1.1.
Output:
    1. $y$ : Public Key
    2. $x$ : Private Key
Process:
    1. $x \leftarrow \mathbf{Z}_q^*$
    2. $y = g^x \bmod p$
    3. Output $(y, x)$
```

Encode

Before we get into encryption, there is a caveat to what an ElGamal cryptosystem in Resist is required to do. Most voting systems advocate for this as well, and the reason becomes clear in Section 6.3 which explains the encryption of ballots.

If an integer plaintext message is beyond the message space $\mathcal{M}$, then it can be encoded into the space by encoding it back into the $\mathcal{M}$. This is done by making the element a discrete log instead of the element itself. Algorithm 6.1.3 presents how this is done.

It is imperative that this encoding remain lossless. Encoded messages as this are used to record vote choices, and this value (which although cannot be decoded) needs to record the plaintext message in a lossless manner.

Algorithm 6.1.3: Encode

```
Due to : Cramer and Shoup [83]
Input  :
    1. $m$ : An integer message, such that $m \in \mathbf{Z}_q^*$
Output:
    1. $M$ : An encoded message, such that $M \in \mathcal{M}$
Process:
    1. $M = g^m \bmod p$
```

Encryption

If a plaintext needs to be encoded into the message space $\mathcal{M}$, this needs to be performed prior to encryption. In this case, the result of the encoding is treated as the message that is encrypted.

The use of a random value (chosen from $\mathbf{Z}_q^*$) ensures that the resulting ciphertext for each encryption of the same plaintext looks different. If an adversary has access to a few encryptions of the plaintext in question, they still fail to ascertain the plaintext corresponding to the ciphertext in question [84].

The resulting ciphertext contains two parts, α and β. α encrypts the value used for randomizing the output. β encrypts the message itself, along with the randomization of the encrypting (public) key.

Re-encryption

Given a ciphertext, the plaintext can be re-encrypted to take a different form. The use of re-encryption will be seen in later sections, in the implementation of Re-encryption Mixnets and Ensuring a correctly cast vote.

A re-encryption of the message does not require any decryption or knowledge of the plaintext. The operation is performed on the ciphertext. The algorithm is presented in Algorithm 6.1.5

Algorithm 6.1.4: ElGamal Encryption

Due to : Taher ElGamal [82]

Input :

1. m : An integer message that needs to be encrypted. This message does not need to be in the message space $\mathcal{M}$.
2. y : Public key y, generated in Algorithm 6.1.2

Output :

1. α : Part of the ciphertext enclosing the randomness, r.
2. β : Part of the ciphertext enclosing the message, m.

Process:

```
1. $M = \text{Encode}(m)$
2. $r \leftarrow \mathbf{Z}_q^*$
3. $\alpha = g^r \bmod p$
4. $\beta = My^r \bmod p$
5. Output($\alpha$, $\beta$)
```

Decryption

The encrypted message will have two parts α and β. The two need to be decrypted down to the correct plaintext regardless of the value of r used for encryption (or re-encryption).

In Algorithm 6.1.6, $\frac{b}{a^x}$ implies that b is multiplied with the inverse of a^x within the finite field p. One of the most common ways of determining the inverse of a value in a finite field is by using Fermat's Little Theorem [85].

A primitive representation of the theorem is as follows.

$a^p \equiv a \pmod{p}$,

where p is a prime number.

Algorithm 6.1.5: ElGamal Re-encryption

Due to : Taher ElGamal [82]

Input :

1. y : Public Key y, generated in Algorithm 6.1.2.
2. $c = (\alpha, \beta)$: Ciphertext obtained from ElGamal Encryption using Algorithm 6.1.4.

Output :

1. $c' = (\alpha', \beta')$: Ciphertext c encrypted under a new randomness r

Process:

```
1. $r \leftarrow \mathbf{Z}_q^*$
2. $\alpha' = \alpha g^r \bmod p$
3. $\beta' = \beta y^r \bmod p$
4. Output$((\alpha', \beta'))$
```

This can further be simplified to the following.

$a^{-1} \equiv a^{p-2} \pmod{p}$

Algorithm 6.1.6 shows a basic method of reducing the ciphertext to the value M. However, if this value was an encoding of the original plaintext, it needs to be decoded. This remains a problem for large numbers, while a few solutions exist for smaller integral plaintext values. The Baby-Step-Giant-Step algorithm [86], Functional Field Sieve [87], and Index Calculus algorithm [88] are some of the common ways of solving the discrete logarithm problem efficiently for smaller values.

Since the ElGamal parameters used for Resist will be 512 bits or greater, traditional approaches fail to efficiently solve the Discrete Logarithm Problem. For example, using the Baby-Step-Giant-Step algorithm requires constructing a large hash table. 512-bit parameters will result in a table whose size exceeds traditional computer memory size.

Since the number of participants in an election will not exceed a few billion, it is more convenient to loop through encodings and determine equality than to create a hash table of the encodings of 512-bit numbers.

```
Algorithm 6.1.6: ElGamal Decryption
  Input   :
    1. $x$ : Private Key $y$, generated in Algorithm 6.1.2.
    2. $c = (\alpha, \beta)$ : Ciphertext obtained from ElGamal Encryption using Algorithm
       6.1.4 or Algorithm 6.1.6.
  Output :
    1. $m$ : Original plaintext integer message, $m$
  Process:
    1. $M = \frac{\beta}{\alpha^x} \bmod p$
    2. $m = \text{Decode}(M)$
    3. Output($m$)
```

6.1.2 Commitments and Hashes

A cryptographic hash is a one-way mathematical function that maps a arbitrary-length plaintext data to a fixed-length bit-string. Being "one-way" implies that the function is irreversible, i.e., the hash cannot be turned back to the original string. Hashes are usually used to sign or verify the integrity of a message/claim [89].

Resist uses hashes in order to generate "commitments". A commitment by an entity or player in the system is a way for them to assert the correctness of a claim. It is used, for example, in Algorithm 6.1.10 to help tellers verify that the key shares being generated for all tellers are correct and all players in the system are honest.

Algorithm 6.1.7: Commitment

```
Input :
    $m$ : integer or string plaintext message.
Output:
    $h$ : integer hash of message $m$.
Process:
    1. $h = hash(m)$
    2. Output $h$
```

6.1.3 Zero Knowledge Proofs

A Zero knowledge proof is a way for one player (the prover, $\mathcal{P}$) in a system to prove to another player (the verifier, $\mathcal{V}$) that they know the contents of an encrypted message, or have knowledge of a value, without revealing the value itself. As a result of verifying the correctness of the proof, $\mathcal{V}$ cannot learn or reveal any new information about the value itself.

The proof KnowDlog, shown in Algorithm 6.1.8, is one example of a zero knowledge proof. Here, the prover $\mathcal{P}$ attempts to prove that they have knowledge of the discrete logarithm, given public input of the generator and an element in the finite field, p. This proof is used in the Distributed Key Generation Algorithm, shown in Algorithm 6.1.10.

Similarly, Algorithm 6.1.9, EqDlogs is derived from a protocol previously proposed for verification of cryptographically secure electronic wallets [91]. It's been used in Resist and Civitas [14] when tabulation tellers need to jointly decrypt a ciphertext using their key shares.

The protocol is used to prove that they are aware of their private key share, x_i, and are using it to decrypt the the given ciphertext. The proof leverages the fact that both, the public key y_i and the decryption share a_i are elements in a cyclic group, whose discrete log is the private key share, x_i.

Algorithm 6.1.8: ⟨Protocol⟩ KnowDlog

Due to: The protocols for Signature Generation and Verification presented by Schnorr [90]

Principals: Prover $\mathcal{P}$ and Verifier $\mathcal{V}$

Public Input:

1. g, y : Parameters to Prove

Private Input:

1. x : Private/Secret Value, such that, $y \equiv g^x (\text{mod } p)$

The Protocol:

1. $\mathcal{P}$: Compute :
    - $z \leftarrow \mathbf{Z}_q^*$
    - $a = g^z \text{ mod } p$
    - $c = hash(g, a) \text{ mod } q$
    - $r = (z + cx) \text{ mod } q$
2. $\mathcal{P} \rightarrow \mathcal{V}$: a, c, r
3. $\mathcal{V}$: Verify $g^r \equiv ay^c (\text{mod } p)$

More examples of Zero Knowledge Proofs will be found in the sections that follow. Resist uses these proofs to make the system verifiable, and help untrusted parties (players) verify the values, credentials, and encrypted messages that they are receiving.

6.1.4 Distributed Cryptosystem

While the votes that are cast to Resist are encrypted by a single public key, the corresponding private key for the system does not lie with a single player or entity. Instead, the system provides shares of the private key to multiple tabulation tellers; these players are chosen with the understanding that colluding with each other is

Algorithm 6.1.9: ⟨Protocol⟩ EqDlogs

Due to: The Basic Signature Scheme for Wallet Databases with Observers, presented by Chaum and Pedersen [91].

Principals: Prover $\mathcal{P}$ and Verifier $\mathcal{V}$

Public Input:

g, a : Two different generators of two different finite fields; The order of both these fields are equal, i.e., p.

p, q : ElGamal parameters that define the aforementioned finite fields.

y_i, a_i : Elements belonging to each of the two finite fields, generated by g and a respectively.

Private Input:

x_i : Private/Secret Value, which serves as the discrete log of elements y_i and a_i in their respective finite fields, i.e., $y_i \equiv g^{x_i} \pmod p$ and $a_i \equiv a^{x_i} \pmod p$

The Protocol:

1. $\mathcal{P}$: Compute :
   - $z \leftarrow \mathbf{Z}_q^*$
   - $u = g^z \pmod p$
   - $v = a^z \pmod p$
   - $w = hash(y_i, a_i, u, v) \bmod q$
   - $r = (z + wx_i) \bmod q$
2. $\mathcal{P} \rightarrow \mathcal{V}$: u, v, w, r
3. $\mathcal{V}$: Verify $g^r \equiv uy_i^w \pmod p$ and $a^r \equiv va_i^w \pmod p$

against their interests.

Distributing the private key results in a system that has a slightly different key generation and decryption algorithm. The principle behind the system remains the same, and so does the way that the encryption is handled.

When keys are generated for a distributed cryptosystem, as shown in Algorithm 6.1.10, each of the private key shares are (uniformly) randomly chosen and made discrete logs of the public key shares. This step is similar to regular ElGamal Key Generation (Algorithm 6.1.2). The overall public key is the product of all public key shares, while each private key share is given out to the respective players (tabulation tellers).

Algorithm 6.1.10: ⟨Protocol⟩ Distributed ElGamal Key Generation

Due To: Distributed ElGamal Key Generation in Civitas [14].

Public Input.

$(p,\ q,\ g)$: ElGamal Key Parameters, generated using Algorithm 6.1.1

n : Number of Shares

Output.

Y : ElGamal Public Key

$[y_1, y_2, ..., y_n\ ]$: Public Key Shares, y_i

$[x_q, x_2, ..., x_n\ ]$: Private Key Shares, x_i

The protocol:

1. S_i : $x_i \leftarrow \mathbf{Z}_q^*$; $y_i = g^{x_i} \bmod p$;
2. S_i : Publish $\mathsf{Commit}(y_i)$
3. S_i : Barrier: wait until all commitments are available
4. S_i : Publish y_i and Proof of $\mathsf{KnowDlog}(g,\ y_i)$
5. S_i : Verify all commitments and proofs
6. $Y = \prod_{i=1}^{n} y_i \bmod p$ is the distributed public key
7. $X = \sum_{i=1}^{n} x_i \bmod p$ is the distributed private key

As mentioned earlier, all plaintext messages are encrypted using the common public key share generated in Step 6 of Algorithm 6.1.10. However, when this ciphertext needs to be decrypted, each of the players use their respective private key shares to compute a part of the decryption. They also provide proofs (EqDlogs) to help verify that they were honest in their partial decryption. The partial decryptions are combined to enable an overall decryption of the value at which point it appears similar to regular ElGamal Decryption. Algorithm 6.1.11 shows how the distributed decryption is handled. This mimics Resist's implementation.

Algorithm 6.1.11: Distributed ElGamal Decryption

Due To: Distributed ElGamal Decryption in Civitas [14].

Public Input.

c : ElGamal Ciphertext, $c = (a, b)$, generated using Algorithm 6.1.4.

$[y_i]$: Public Key Shares, $[y_1, y_2, ..., y_n]$, corresponding to the private keys (x_i) of each of the n tabulation tellers.

Private Input.

$[x_i]$: Private Key Shares, $[x_1, x_2, ..., x_n]$, kept secret by each of the n tabulation tellers.

Output.

M : Plaintext message encrypted by the ciphertext c

The protocol:

1. S_i : Publish share $a_i = a^{x_i} \bmod p$ and proof EqDlogs(g, a, y_i, a_i).
2. S_i : Verify all proofs.
3. $A = \prod_{i=1}^{n} a_i \bmod p$
4. $M = \frac{b}{A} \bmod p$
5. Output M.

6.1.5 Homomorphic Encryption

Homomorphic Encryption is a subset of different encryption types. A cyptosystem that is homomorphic allows for computations to be performed on ciphertexts in such a way that the result of the computation is an encryption of what the plaintext result of the same computation would have been [92].

To put it in simpler words, homomorphic public key cryptosystems, allow for either the addition or the multiplication of their ciphertexts, the results of which can decrypt to something meaningful.

While this is possible with RSA, it does not satisfy the level of security required for a voting system since raw RSA isn't IND-CPA secure [33]. If the cryptosystem is susceptible to plaintext attacks, an adversary can determine the value of a particular vote choice with the help of a chosen-plaintext attack. ElGamal, on the other hand, has ciphertexts that are a tuple, $c = (g^r, m.y^r)$. A homomorphic operation on this is similar to

$(g^{r1}, m_1.y^{r1}) \otimes (g^{r2}, m_2.y^{r2}) = (g^{r1+r2}.(m_1m_2).y^{r1+r2})$.

Traditional ElGamal, as seen above, is homomorphic in multiplication when the plaintext message m is in the base. Since voting systems require an addition of messages (votes), this can be achieved if the message is raised to the exponent, as is done in Encoding (shown in Algorithm 6.1.3).

The resulting homomorphic addition looks as follows. This has been previously described in their thesis by Ben Adida [33].

$$\begin{aligned} & \mathcal{E}_{pk}(m_1; r_1) \otimes \mathcal{E}_{pk}(m_2; r_2) \\ = \ & (g^{r1}, g^{m_1} y^{r_1}) \otimes (g^{r2}, g^{m_2} y^{r_2}) \\ = \ & (g^{r_1+r_2}, g^{m_1+m_2} y^{r_1+r_2}) \\ = \ & \mathcal{E}_{pk}(m_1 + m_2; r_1 + r_2). \end{aligned}$$

Using homomorphic encryption is one way to allow for ballot privacy. It ensures that an individual vote is never decrypted and hence the vote cast by a single voter is never known. The result is an overall tally that can be computed and verified, and ballot secrecy remains despite the results having been evaluated. It is often used as an alternative to Mixnets (described next), and an implementation of this was seen in the first version of Helios [52].

6.1.6 Mixnets

An alternative to using traditional homomorphic encryption is the use of mixnets. Mixnets involve multiple parties (tabulation tellers in this case) shuffling a list of ciphertexts in such a manner that the order of the resulting output is randomized and reencrypted. The shuffled output from one party serves as the input to the following party.

While there are robust implementations of mixnets that can substitute the one described in this thesis, we introduce the Jackobsson, Juels, and Rivest [93] Random Partial Checking (RPC) implementation of it. The algorithms used give a primitive overview of its implementation, leaving out details for needed for robustness.

Algorithm 6.1.12 shows the shuffle performed on a single server, by a single party. They receive a list of ciphertexts, L, and the direction, dir, for which proofs are required to be generated. Each element in the list is shuffled randomly, and reencrypted on shuffle so that the resulting list, L_R, cannot be mapped back to the input.

Proof of correct mixes are generated with a primitive approach. Depending on whether the dir is towards the input or the output, the corresponding permutation of the shuffle is added to the commitment. Each commitment also includes a witnessm w. The witnesses are random bitstrings taken from the Universal domain, which can include elements beyond $\mathbf{Z}_q^*$.

A commitment is made for each element in the mix. The result is an additional list L_C, which has an equal number of corresponding elements as L_R. Both of these list are sent as an output to the next mix server.

Protocol 6.1.13 gives an overview of how the list of ciphertexts, L, passes through each server performing Algorithm 6.1.12, and gets verified. Each party (tabulation teller) involved in the mix, shuffles and reencrypts the ballot twice before the shuffled list on to the next party. In the first mix, they produce commitments for the shuffle in the output direction, out, and in the second mix, they produce commitments for the shuffle in the input direction, in. They additionally choose a random bitstring q_i and publish a commitment to it.

Once all the shuffles have been completed, random bitstrings, q_i's, are combined and hashed to render a single random value, $\mathcal{Q}$. This value, $\mathcal{Q}$, is used to obtain random bits $\in \{0, 1\}$ to verify commitments from each party in one direction $\in$

Algorithm 6.1.12: Mix

Due to : Randomized Partial Checking (RPC) proposed by Jackobsson, Juels, and Rivest [93].

Input :

1. List L : List of ciphertexts that need to be mixed of length m, $L = [c_1,\ c_2,\ ...,\ c_m]$
2. dir : Direction of Verification, such that, $dir \in \{in, out\}$.

Output :

1. List L_R : Reencrypted and mixed ciphertext list.
2. List L_C : List of commitments over the space of permutations, the direction of which is determined by dir

Process:

```
1. $\pi \leftarrow$ Space of permutations over m elements
2. If $dir = in$ then $p = \pi$ else $p = \pi^{-1}$
3. $r_1 \leftarrow \mathbf{Z}_q^*;\ ...; r_m \leftarrow \mathbf{Z}_q^*;\ w_1 \leftarrow \mathcal{O};\ ...;\ w_m \leftarrow \mathcal{O}$
4. $L_R = [\mathsf{Reenc}(c_{\pi(1)}; r_1),\ ...,\ \mathsf{Reenc}(c_{\pi(m)}; r_m)]$
5. $L_C = [\mathsf{Commit}(w_1, p(1)),\ ...,\ \mathsf{Commit}(w_m, p(m))]$
6. Output $L_R,\ L_C$.
```

$\{in, out\}$. If all commitments are correct (despite random challenges), the output of the second mix of the n^{th} party is the output of the entire mixnet.

The algorithm for the mixnet is not robust. If a commitment from one teller is incorrect, then the entire mixnet computation is discarded and run from the initial mix, eliminating the coerced tabulation teller.

6.2 Managing Credentials in Resist

This section gives an overview of how credentials are generated, verified, encrypted, and managed by Resist. The algorithms defined in this section give an understanding of the principles behind credential management, but the implementation (described in the following chapter) shows how they look from the perspective of the end-user.

6.2.1 Registration

Each authenticated voter in Resist participates in an election only after registering for it. During authentication, they receive two keys; an authentication key and a registration key. These keys are used to interact with the registration tellers for an election and obtain their voting credential.

When creating an election, the supervisor assigns multiple users of the system as registration tellers for the specific election. Each of these tellers can provide only a share of the credential to the voters that they authenticate. They additionally encrypt these shares (with the election's public key) and add them to the bulletin board.

Credential Generation

Each registration teller for an election provides a credential share, s_i, to an authenticated voter. These shares are random values that are chosen from the message space, $\mathcal{M}$.

$s_i \leftarrow \mathbf{Z}_q^*$

These shares are multiplied to produce the overall credential for each voter. Beside the voter, no player in the system can look at the Credential itself. If there are n credential shares, the Credential is calculated as $S = \prod_{i=1}^{n} s_i$.

Algorithm 6.1.13: ⟨Protocol⟩ MixNet

Due To: Randomized Partial Checking (RPC) proposed by Jackobsson, Juels, and Rivest [93].

Principals: Tabulation Tellers $\mathrm{TT}_1, \ ..., \ \mathrm{TT}_n$

Public Input.

1. List L : List of ciphertexts that need to be mixed of length m, $L = [c_1, \ ..., \ c_m]$

Output. $\mathsf{Mix}_{n,2}.L_R$: Anonymized list of ciphertexts.

The protocol:

```
1. Let $\mathsf{Mix}_{i,j}$ denote the $j^{th}$ mix performed by the $i_{th}$ teller.
   Initialize the first mix input as $\mathsf{Mix}_{0,2}.L_R = L$
2. For i = 1 to $n$, sequentially:
   (a) $\mathrm{TT}_i$: Let $L_1 = \mathsf{Mix}_{i-1,2}.L_R$; Publish $\mathsf{Mix}_{i,1} = \mathsf{Mix}(L_1, out)$
   (b) $\mathrm{TT}_i$: Let $L_2 = \mathsf{Mix}_{i,1}.L_R$; Publish $\mathsf{Mix}(L_2, in)$
   (c) $\mathrm{TT}_i$: $q_i \leftarrow \mathcal{O}$; Publish $\mathsf{Commit}(q_i)$
3. All $\mathrm{TT}_i$: Publish and verify $q_i$; Let $\mathcal{Q} = hash(q_1, \ q_2, \ ..., \ q_n)$
4. All $\mathrm{TT}_i$:
   (a) Let $\mathcal{Q}_i = hash(\mathcal{Q}, i)$
   (b) For all $j \ in \ [1, \ ..., \ m]$, publish $r_j$, $w_j$, and $p(j)$ from $\mathsf{Mix}_{i,1+\mathcal{Q}_i[j]}$
   (c) Verify all commitments ( witnesses, $w$ and permutations, $p$) and
       reencryptions $r$ from all other tellers, i.e.:
       i. Verify $w_j$ and $p(j)$ against $\mathsf{Mix}_{i,1+\mathcal{Q}_i[j]}.L_C[j]$
       ii. If $\mathcal{Q}_i[j] = 0$ then verify: $\mathsf{Reenc}(\mathsf{Mix}_{i-1,2}.L_R[p(j)]; r_j) = \mathsf{Mix}_{i,1}.L_R[j]$,
           else if $\mathcal{Q}_i[j] = 1$ then verify: $\mathsf{Reenc}(\mathsf{Mix}_{i,1}.L_R[j]; r_j) = \mathsf{Mix}_{i,2}.L_R[p(j)]$
5. Output $\mathsf{Mix}_{n,2}.L_R$.
```

Credential Encryption

Before we begin, we need to explain the concept of malleability. When votes are encrypted in an ElGamal-based, homomorphic operations can be performed on them, making them malleable. Sine credentials are calculated by performing a multiplication operation on their shares, any entity that has access to the bulletin board with encrypted credential shares should not be capable of casting a vote with an encrypted credential.

The tuple (γ, θ) turns the encryption non-malleable. It is called a Schnorr signature [90]. The Credential Encryption Algorithm 6.2.1 is a variation of conventional non-malleable encryption in that the randomization is provided, voter and registration teller identifiers are included in the hash, and the credential share s_i does not need to be encoded into $\mathcal{E}$.

Credential Verification

Prior to decrypting any non-malleable encryption, and in this case the credential encryption, the Schnorr Signature needs to be verified. Algorithm 6.2.2 shows how this is done. It can also be implemented by any observer who can access Resist's bulletin board to verify that encryptions have been performed correctly.

Digital Verifier Reencryption Proof

When a voter attempts to obtain their credential, the registration teller encrypts the generated credential with the election public key and adds it to the bulletin board. This encrypted value is used by the tabulation tellers to check the validity of votes.

The registration teller is tasked with proving to the voter that the credential share that has been sent to the bulletin board is the same as the share that they have received. The proof is handled by reencrypting the encrypted share, and passing along a zero-knowledge proof that both of these correspond to the same plaintext. Hirt and Sako [51] used this as one of the proofs for re-encryption in their homomorphic voting scheme. Employing the Protocol 6.2.3 helps generate a slightly different algorithm, $\mathsf{Fake\ DVRP}$, to aid with coercion resistance.

Algorithm 6.2.1: Credential Encryption

Due to : Credential Encryption, derivation of Non-Malleable Encryption from Civitas [14].

Input :

K_{TT} : ElGamal Public Key

s : Credential Share, such that $s \in \mathcal{M}$

r : Randomization Factor

rid : Identifier of the Registration Teller

vid : Identifier of the Voter

Output:

$\mathcal{E}(s)$: Encrypted credential share, such that, $\mathcal{E}(s) = (\alpha, \beta, \gamma, \theta)$

Process:

```
1. $t \leftarrow \mathbf{Z}_q^*$
2. $(\alpha, \beta) = \mathsf{Enc}(s; r; K_{TT})$
3. $\gamma = hash(g^t \pmod p),\ \alpha,\ \beta,\ rid,\ vid) \bmod q$
4. $\theta = (t + \gamma r)\mathrm{mod} q$
5. Output $(\alpha, \beta, \gamma, \theta)$
```

Algorithm 6.2.2: Credential Verification

```
Due to : Schnorr [90]
Input :
    $\mathcal{E}(s)$ : Encrypted credential share, such that, $\mathcal{E}(s) = (\alpha, \beta, \gamma, \theta)$
    $rid, vid$ : Identifiers of the voter and the registration teller
Output:
    $< boolean >$ : True/False depending on the result of the verification
Process:
    1. $V \;=\; hash(g^{\theta} a^{-\gamma} \text{ (mod } p),\ \alpha,\ \beta,\ rid,\ vid) \text{ mod } q$
    2. Output $V \;==\; \gamma$
```

Obtaining Credential Shares

In order to obtain credential shares, the voter interacts with all registration tellers. Each registration teller authenticates them using their authentication key. Once the voter has been authenticated, they provide them an encrypted credential share, a different encryption of which is posted to the bulletin board. They also provide a DVRP Proof that the value posted on the bulletin board is the same as the value that has been sent to the voter. The entire protocol can be found in Algorithm 6.2.4.

In order to authenticate each other, the voter and the registration teller, run a version of Needham-Schroeder-Lowe [75, 76], by generating nonces for verification. They also jointly agree upon a symmetric (AES) secret session key for sending an receiving the credential share and associated proofs.

6.2.2 Fake Credentials

Once a voter has received the credential shares, they can decrypt these shares, verify the associated DVRP, and combine them to obtain their credential. Before they destroy their credential shares, however, it will be useful for them to generate fake credentials to be used in the event that they get coerced.

Algorithm 6.2.3: ⟨Protocol⟩ Digital Verifier Reencryption Proof

Due to : Hirt and Sako [51]

Principals : $[\mathcal{P}]$: Prover (Registration Teller) and $[\mathcal{V}]$: Verifier (Voter)

Public Input :

K_{V_A} : Public key of $\mathcal{V}$

e : ElGamal Ciphertext $e = (\alpha, \beta)$

e' : Reencrypted ElGamal Ciphertext, $\ni e' = \mathsf{Reenc}(e; r'; K_{TT}) = (\alpha', \beta')$

K_{TT} : Public key under which e and e' are encrypted

Private Input:

ζ : witness, such that, $\alpha' \equiv \alpha g^{\zeta} \pmod p$ and $\beta' \equiv \beta K_{TT}^{\zeta} \pmod p$

Output : $(c == c')$

Process :

```
1. $\mathcal{P}$: Compute:
   (a) $d,\ w,\ r \leftarrow \mathbf{Z}_q$
   (b) $a = g^d \pmod p$; $b = K_{TT}^d \pmod p$; $s = g^w K_{V_A}^r \pmod p$
   (c) $c = hash(e,\ e',\ a,\ b,\ s) \pmod q$
   (d) $u = (d + \zeta(c + w)) \pmod q$
2. $\mathcal{P} \rightarrow \mathcal{V}$: $c,\ w,\ r,\ u$
3. $\mathcal{V}$: Compute:
   (a) $a' = g^u/(\alpha'/\alpha)^{c+w} \pmod p$; $b' = K_{TT}^u/(\beta'/\beta)^{c+w} \pmod p$; $s' = g^w K_{V_A}^r \pmod p$
   (b) $c' = hash(e,\ e',\ a',\ b',\ s') \pmod q$
4. $\mathcal{V}$: Output $(c == c')$
```

Algorithm 6.2.4: ⟨Protocol⟩ Registering for an Election

Due to : Steps 1 to 8: Needham-Schroeder-Lowe [75].
Steps 9 to 11: Civitas' adaptation of [94, 51, 12]

Principals : $[RT_i]$: Registration Teller and V : Voter

Public Input :

1. K_{TT}, K_{RT_i} : Public keys of the Election and RT_i, respectively.
2. K_{V_D}, K_{V_A} : Voter's public designation key and authentication key, respectively.
3. eid, vid, rid : Identifiers of the election, V, and RT_i, respectively.
4. S_i : Encrypted Public Credential, $S_i = \mathsf{CredEnc}(s_i; r; K_{TT}; rid; vid)$

Private Input: RT_i : Private Credential,$s \in \mathcal{M}$; randomization factor, $r \in \mathbf{Z}_q^*$; $[V]$: Private Authentication Key, k_{V_A}

The Protocol :

1. V: $N_V \leftarrow \mathcal{N}$, where $\mathcal{N}$ is the space for nonces.
2. $V \rightarrow RT_i$: $\mathsf{Enc}(eid, vid, N_V; K_{RT_i})$
3. RT_i: Verify eid, vid and all other credential shares posted by other talliers, i.e., for all j, S_j exists and $\mathsf{CredVer}(S_k; rid_j; vid)$ returns true.
4. RT_i: $N_R \leftarrow \mathcal{N}; k \leftarrow \mathsf{Gen}_{AES}(1^l)$, where l is the security parameter for AES
5. $RT_i \rightarrow V$: $\mathsf{Enc}(rid, N_R, N_V, k; K_{V_A})$
6. V: Verify rid and N_V
7. $V \rightarrow RT_i$: N_R
8. RT_i: Verify N_R; $r' \leftarrow \mathbf{Z}_q^*; \zeta = r' - r; S_i' = \mathsf{Enc}(s_i; r'; K_{TT})$
9. $RT_i \rightarrow V$: $\mathsf{Enc}_{AES}(s_i, r', \mathsf{DVRP}(K_{V_D}, S_i, S_i'; \zeta); k)$
10. V: Verify $S_i' = \mathsf{Enc}(s_i; r'; K_{TT})$, and verify DVRP against S_i from the bulletin board.

Creating Fake Credentials

To create fake credentials, the voter needs to be certain of the set of registration tellers that they trust to be honest. The shares obtained from these tellers will be replaced and the those obtained from all other teller will be retained. This is done to ensure that in the event that any of the non-trusted registration tellers colludes with the coercer and passes the their part of the voter's credential share on, the coercer finds the share that they are aware of to be unaltered.

For every new credential share that is made replacing the old share, a new Fake DVRP needs to be created as shown in Algorithm 6.2.5. The voter has an advantage of knowing all credential shares and can therefore leverage it to make the new credential appear to be verifiable.

Resist, like Civitas, assumes that at least one registration teller is known by the voter to be honest. While this is an improvement from the JCJ Protocol which required a majority of the tellers to be honest, it still remains a caveat, the absence of which can be detrimental to the coercion resistance of the system.

Additionally, the coercer may ask an authenticated voter to abstain from participating in registration. This is a hinderance towards satisfying all properties that make the system coercion resistant. Making a voter abstain from registering makes the coercion public, since all participating entities can identify that the voter did not participate in the registration process.

6.3 Casting a Vote

When the supervisors creates an election, they post a list of ciphertexts on the bulletin board. This list contains all accepted ciphertexts of candidate choices. If a voter wishes to mark a ballot for a particular candidate, they merely re-encrypt the corresponding ciphertext from this list.

Resist employs re-encryption to ensure that the list of ciphertexts are restricted, and a coercer cannot make a voter launch a randomization attack. The voter also needs to send a proof (a 1-out-of-L Proof) that the encrypted choice is merely a re-encryption of the elements from the list. This protocol is shown in Algorithm 6.3.1.

The Vote Proof (Algorithm 6.3.2) shows the well formedness of the vote itself. It

Algorithm 6.2.5: ⟨Protocol⟩ Fake DVRP

Due to : Hirt and Sako [51].

Principals : Prover, $\mathcal{P}$ and Verifier, $\mathcal{V}$

Public Input :

K_{V_D} : Public key of $\mathcal{P}$

e : ElGamal Ciphertext $e = (\alpha, \beta)$

e' : Fake ElGamal Ciphertext, such that, $\tilde{e} = (\tilde{\alpha}, \tilde{\beta})$

K_{TT} : Public key under which e and e' are encrypted

Private Input: k_{V_D}, Private key of $\mathcal{P}$

Output : $(c == c')$

The Protocol:

1. P: Compute:
   (a) $\gamma, \theta, \tilde{u} \leftarrow \mathbf{Z}_q^*$
   (b) $\tilde{a} = g^{\tilde{u}}/(\tilde{\alpha}/\alpha)^{\gamma} \pmod p$
   (c) $\tilde{b} = h^{\tilde{u}}/(\tilde{\beta}/\beta)^{\gamma} \pmod p$
   (d) $\tilde{s} = g^{\theta} \pmod p$
   (e) $\tilde{c} = hash(e, \tilde{e}, \tilde{a}, \tilde{b}, \tilde{s}) \bmod q$
   (f) $\tilde{w} = \gamma - \tilde{c} \pmod q$
   (g) $\tilde{r} = (\theta - \tilde{w})/k_{V_D} \pmod q$
2. $\mathcal{P} \rightarrow \mathcal{V}$: $\tilde{c}, \tilde{w}, \tilde{r}, \tilde{u}$
3. $\mathcal{V}$: $\tilde{c}, \tilde{w}, \tilde{r}, \tilde{u}$ will be verified as DVRP, shown in Algorithm 6.2.3.

Algorithm 6.2.6: Fake Credential

Due to : $\mathsf{FakeCred}$ as used in Civitas [14].

Input :

s_i : List of Private Credential Shares

S_i : List of Public Encrypted Credential Shares

r_i : Randomization Factors

D_i : List of DVRP's from each teller, RT_i

L : List of honest registration tellers

$(K_{V_D}, k_{V_D}$: Public and Private Designation Keys for the Voter, V

Output :

$\tilde{s_i}$: Fake credential shares

$\tilde{r_i}$: Fake randomization factors

$\tilde{D_i}$: Fake DVRP's

The Process:

```
1. For all $i \in L$:
    (a) $\tilde{r_i} \leftarrow \mathbf{Z}_q^*$; $\tilde{s_i} \leftarrow \mathcal{M}$; $\tilde{S_i} = \mathsf{Enc}(\tilde{s_i}; \tilde{r_i}; K_{TT})$
    (b) $\tilde{D_i} = \widetilde{\mathsf{DVRP}}(K_{V_D}, S_i, \tilde{S_i}; k_{V_D})$
2. For all $i \notin L$:
    (a) $\tilde{r_i} = r_i$; $\tilde{s_i} = s_i$; $\tilde{S_i} = \mathsf{Enc}(s_i; r_i; K_{TT})$
    (b) $\tilde{D_i} = \widetilde{DVRP}(K_{V_D}, S_i, \tilde{S_i}; k_{V_D})$
3. Output $(\tilde{s_i}, \tilde{r_i}, \tilde{D_i})$ for all $i$
```

Algorithm 6.3.1: ⟨Protocol⟩ Reenc Proof

Due to : Hirt and Sako [51] and Juels et al. [12].

Principals : Prover $\mathcal{P}$ and Verifier $\mathcal{V}$

Public Input :

C : List of L ciphertexts, such that, $C = \{(\alpha_i, \beta_i) | 1 \leq i \leq L\}$

c : ciphertext, such that, $c = (\alpha, \beta)$

Private Input: $t \in [1...L]$, $r \in \mathbf{Z}_q^* \ni (\alpha, \beta) = (g^r \alpha_t \pmod p), y^r \beta_t \pmod p)$)

Output : $(c' == D')$

The Protocol :

1. $\mathcal{P}$: Compute:
   (a) $d_i, r_i \leftarrow \mathbf{Z}_q^*$, such that, $i \in [1...L]$
   (b) $\{(a_i, b_i) | i \in [1...L]\}, where : a_i = (\frac{\alpha_i}{\alpha})^{d_i} g^{r_i} \pmod p; \; b_i = (\frac{\beta_i}{\beta})^{d_i} y^{r_i} \pmod p;$
   (c) $E = \alpha, \beta, \alpha_1, ..., \alpha_L, \beta_1, ...\beta_L$; $c = hash(E, a_1, ..., a_L, b_1, ..., b_L) \bmod q$
   (d) $w = (-rd_t + r_t) \bmod q$
   (e) $D = c - (\sum_{i \in [1...t-1, t+1, ..., L]} d_i) \bmod q$
   (f) $R = (w + rd'_t) \bmod q$
   (g) $d_i^\beta = \begin{cases} d_i : & i \neq t \\ D : & i = t \end{cases}$
   (h) $r_i^\beta = \begin{cases} r_i : & i \neq t \\ R : & i = t \end{cases}$
2. $\mathcal{P} \rightarrow \mathcal{V}$: $(d_1^\beta, ...d_L^\beta, r_1^\beta, ...r_L^\beta)$
3. $\mathcal{V}$: Compute:
   $\{(a_i^\beta, b_i^\beta) | i \in [1...L]\}$, computed similar to step 1.(a), but with d_i^β and r_i^β replacing d_i and r_i;
   $c' = hash(E, a^v\beta_1, ..., a_L^\beta, b_1^\beta, ..., b\beta_L) \bmod q$;$D' = \sum_{i \in [1...L]} d_i^\beta \bmod q$
4. $\mathcal{V}$: Output $\mathsf{ReencPf} = (c' == D')$

encapsulates the context of the election, the marked choices, the encrypted credential, and knowledge of the public index the vote has been cast against.

Casting a vote is simply a re-encryption of the chosen elements from a public list of accepted ciphertexts. After associating this re-encryption with an encryption of the credential and index, proofs are generated for both, the choices, and the vote as a whole. The vote is then cast directly to the bulletin board. Casting these votes to ballot boxes, as is done, in Civitas will be favourable towards the robustness and availability of the system. Algorithm 6.3.3 demonstrates how a vote is formed and cast on the bulletin board.

6.4 Tabulation

Once all votes have been cast and the voting for the election has been stopped, all tabulation tellers can begin to compute the final tally. Before we introduce the protocol for tabulation, it'll help to introduce one last algorithm, PET.

A Plaintext Equivalence Test, PET, as shown in Algorithm 6.4.1, is used to evaluate if two ciphertexts encrypt the same plaintext message. It does so without decrypting them to their actual value.

Performing a PET requires that the cryptosystem permit homomorphic operations on ciphertexts. It divides the parameters of the two ciphertexts with each other. It then decrypts the result to check the result of this division. If the result decrypts to a value of 1, both the ciphertexts are considered to encrypt the same plaintext.

The Protocol and process for tabulation is described in Algorithm 6.4.2. The process is carried out jointly by all Tabulation Tellers in the system. The results, along with associated proofs, are posted on the bulletin board, endorsed by the Supervisor of the election. Since the protocol has been described verbatim, we direct the reader to refer Algorith 6.4.2 for an explanation of the process.

Algorithm 6.3.2: $\langle$**Protocol**$\rangle$ Vote Proof

Due to : Camenisch and Stadler [58]

Principals : Prover $\mathcal{P}$ and Verifier $\mathcal{V}$

Public Input :

1. (α_1, β_1) : Encrypted Credential
2. (α_2, β_2) : Encrypted Choice
3. (α_3, β_3) : Encrypted Public Index
4. ctx : Vote context, including election identifier (eid), etc.

Let $E = (g, \alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3, \beta_3, ctx)$

Private Input: $a_1, a_2, a_3, suchthat, \alpha_i \equiv g^{a_i} \pmod{p}$

The Protocol :

1. $\mathcal{P}$: Compute:
   (a) $r_1, r_2, r_3 \leftarrow \mathbf{Z}_q$
   (b) $c = hash(E, g^{r_1} \bmod p, g^{r_2} \bmod p, g^{r_3} \bmod p) \bmod q$
   (c) $s_1 = (r_1 - ca_1) \bmod q$
   (d) $s_2 = (r_2 - ca_2) \bmod q$
   (e) $s_3 = (r_3 - ca_3) \bmod q$
2. $\mathcal{P} \rightarrow \mathcal{V}$: c, s_1, s_2, s_3
3. $\mathcal{V}$: Compute $c' = hash(E, g^{s_1} g^{ca_1}, g^{s_2} g^{ca_2}, g^{s_3} g^{ca_3}) \bmod q$
4. Ouput $c == c'$

```
Algorithm 6.3.3: ⟨Protocol⟩ Vote
  Due to       : Juels et. al. [12]
  Principals   : Voter $V$, Bulletin Board $BB$
  Public Input :
    $K_{TT}$ : Election Public Key
    $ctx$ : Vote context, including election identifier ($eid$), et al.
    $C$ : List of $L$ ciphertexts, such that, $C = \{(\alpha_i, \beta_i) | 1 \leq i \leq L\}$
  Private Input:
    $s$ : Credential
    $c_t$ : Candidate choice, such that, $t \in [1...L]$
    $in$ : Index
  The Protocol:
    1. $V$: $a_1 \leftarrow \mathbf{Z}_q^*$; $es = \mathsf{Enc}(s; r_s; K_{TT})$
    2. $V$: $a_2 \leftarrow \mathbf{Z}_q^*$; $ei = \mathsf{Enc}(in; r_i; K_{TT})$
    3. $V$: $a_3 \leftarrow \mathbf{Z}_q^*$; $ev = \mathsf{Reenc}(c_t; r_v)$
    4. $V$: $P_w = \mathsf{VotePf}((\alpha_1, \beta_1) = es, (\alpha_2, \beta_2) = et, (\alpha_2, \beta_2) = ei, ctx, a_1, a_2, a_3)$
    5. $V$: $P_k = \mathsf{ReencPf}(C, ev, t, r_v)$
    6. $V$: vote $= (es, ev, ei, P_w, P_k)$
    7. $V \rightarrow BB$: vote
```

Algorithm 6.4.1: ⟨Protocol⟩ Plaintext Equivalence Test

Due to : Jakobsson and Juels [95]

Principals : Tabulation Tellers, TT_i

Public Input :

c_j : ciphertext of m_j, such that, $\mathsf{Enc}(m_j; K_{TT}) = (\alpha_j, \beta_j)$ for $j \in \{1, 2\}$

Private Input:

x_i : Private Key Share of the Tabulation Teller

Let $R = (\delta, \epsilon) = (\alpha_1/\alpha_2, \beta_1/\beta_2)$

Output : $m_1 == m_2$

The Protocol:

```
1. TT_i: $z_i \leftarrow \mathbf{Z}_q^*; (\delta_i, \epsilon_i) = (\delta^{z_i}, \epsilon^{z_i})$
2. TT_i: Publish $\mathsf{Commit}(\delta_i, \epsilon_i)$; Wait until all commitments have been published.
3. TT_i: Publish $(\delta_i, \epsilon_i)$ and $\mathsf{EqDlogs}(\delta, \epsilon, \delta_i, \epsilon_i)$; Verify all commitments and proofs.
4. Let $c' = (\prod_i \delta_i, \prod_i \epsilon_i)$
5. All TT: m' = $\mathsf{DistDec}(c')$ (Algorithm 6.1.11)
6. Output m' == 1
```

Algorithm 6.4.2: ⟨Protocol⟩ Tabulate

Due to : Spycher et al. [21] and Clark et. al. [14]

Principals : Supervisor *Sup*, Tabulation Tellers TT_1,...,TT_n, Bulletin Board BB

Public Input : Public Election Key K_{TT} and all cast votes (A_i, B_i, C_i) with proofs from the bulletin board

Private Input: Private key share x_i of each teller TT_i, Common random value z shared among all TT_i

Output : Results of the election

The Protocol:

1. **Retrieve Votes and Check Proofs.** Retrieve all votes from BB. Let these votes be V_0. Retrive the list of encrypted credentials, S from BB. Verify VotePf and $\mathsf{ReencPf}$ for each vote. The votes for whom the proofs return true are considered a tuple, (A_i, B_i, C_i) where A_i is the encrypted credential, B_i is the encrypted choice, and C_i is the encrypted index.

2. **Add Random Votes and Anonymize.** Add a random number, X_i, of votes associated with each entry on the voter roll on BB. This resulting list is V_1. Run $\mathsf{MixNet}(V_1)$. Let the anonymized list be $\hat{V_1}$

3. **Eliminate Duplicates and Mix.** Raise all $\hat{A_i}$ to the power of z, $\hat{A_i}^z$ and decrypt down to the blinded credential, a_i^z. Add a_i^z to a hash table and silently drop collisions. The list of unique votes here is V_2, which contains tuples, $(\hat{A_i}, \hat{B_i}, \hat{C_i})$. The list S contains encrypted credentials, σ_j.

4. **Eliminate Invalid Credentials.** The value of $\hat{C_i}$ is decrypted for each tuple in $\hat{V_2}$. The resulting *index* value is used to find the credential from S and replace each vote tuple as $(\hat{A_i}, \hat{B_i}, \sigma_j)$. This tuple is passed through a second reencryption mixnet, anonymizing it to $(\tilde{A_i}, \tilde{B_i}, \hat{\sigma_j})$ The result of $\mathsf{PET}(\tilde{A_i}, \hat{\sigma_i})$ is used to determine if votes are eliminated. The resulting list of votes with valid credentials is V_3.

5. **Decrypt and Tally** All B_i in V_3 are decrypted and tallied. The results are published on BB and *Sup* will endorse the tally.

Chapter 7

System Implementation

Once the algorithms that were required for this protocol to be implemented had been laid out, it helped clarify the requirements for implementing an example system. The web-based application was implemented using the Django Framework. The backend of the system was written in Python and the front-end used Vanilla Javascript to support simple HTML pages. The database that managed user data and the bulletin board was implemented using PostgreSQL.

The sections that follow unpack these components and also present Entity Relationship Diagrams and screenshots from the application.

7.1 The Django Framework

The code for the system was implemented using Django [96]. As a high-level, Python-based web framework, Django simplifies development, includes inherent modularity, and encourages development of code that can have a longer lifecycle. Since the framework has been in active development for a decade and a half at this point, it has comprehensive documentation, guarantees code stability for the foreseeable future, and has a strong, supportive community built around it.

A Django project comprises multiple applications (apps), each of which behave like a separate entity. Each Django app consists of its own "models.py", "urls.py", and "views.py" along with its own "templates" directory. These files are explained below in this section. By sandboxing components of the project into apps, Django allows for an app to serve as a modular, portable section that can be extracted and used in more than just one Django project.

Django presents a version of the Model-View-Controller (MVC) architecture. It provides for separate files for interacting with one of its many relational databases through the"model". Django offers the ability to make HTML templates that take custom tags, embedded python logic, and syntax that can include variables that has been passed to it through another file; this is considered analogous with the "view". Finally, Django handles requests by passing them to a file that interacts with the "model" and passes values to the "view" (template), analogous with the "controller".

It provides its own Object Relational Mapping (ORM), called the "model". The model defines tables and fields in the database with the help of classes and their component properties. Since the user defines these classes in familiar python syntax, the actual implementation of the database remains abstracted from the developer. Django offers an SQLite database by default, but the framework supports MySQL, PostgreSQL, and Oracle as well.

All GET and POST requests are initially parsed through a list of accepted url patterns (defined with the help of regular expressions) that are centrally held in the "urls.py" file. Each incoming request (accepted url pattern) is associated with a particular class or method that handles the request. These handlers are located in the "views.py" file, which in this case is synonymous with the aforementioned controller. These methods can access the "model" and depending on the incoming request, create, retrieve, update, and delete model objects. They also return values through HTTP and JSON responses, redirect to another URL, or render a new template.

The templates folder in any given Django app is populated with HTML files. Django provides for these files to include tags ("{%%}" or "{{}}") that can contain python code, django-specific rendering of variables, static files, et al. Static files associated with templates can include any local files including CSS, Javascript, and Image files.

Using the Django framework for implementing the web-application can help extend available python code to more than one front-end. It can include a REST-framework to handle GET, PUT, POST, and DELETE requests via api calls from other applications. Django can include extended support for the Node Package Manager (npm), which extends to a number of possible Javascript-based front-end framework offerings including React, ReactNative, and AngularJS.

7.2 PostgreSQL as a Database

PostgreSQL is an open-source, object-relational database that is a direct descendant of the Ingres Database developed at the University of California, Berkley [97]. Initially referred to as Postgres95, its name was later changed to PostgreSQL in late 1996.

Given the more commonly supported options by Django, including SQLite and MySQL, PostgreSQL stands out because it is an Object-Relational Database Management System (ORDBMS). In comparison to a conventional RDBMS, it serves as a logical extension of Object-Oriented Programming. It uses classes and inheritance as a means of implying entity relationship, which works in tandem with Django's ORM offering. A Primary Key identifies a "record" in a table (analogous to a single row), and inheritance is managed through a reference field (Foreign Key).

Postgres' support for JSON fields, Array Fields, Slug Fields, and Image/File Fields covers all possible database field types that are needed by a voting system coupled with User Accounts. In the case of Resist, the JSON and Array Fields are particularly used to store ElGamal Keys and Key Parameters that contain long integral numbers. These numbers, however, are first individually converted to strings since Postgres does not recognize extremely long integer numbers as integers.

7.3 Languages

Implementing Resist's protocol required the use of a front-end and back-end web interface. Given the defaults supported by either languages, those chosen were intentionally kept as basic and independent from imported libraries and modifications as possible. This section elaborates on the language and library choices that were incorporated into the code that was developed.

Python

A high-level programming language that is interpreted (instead of compiled), Python is the default language supported by the Django Framework, and also one that the framework has been built upon [98]. Python was made with the intention of making general purpose programming easy to learn, and as a result, it currently has large community support and a considerable number of developers using it [99].

Although Python allows for variables to be loosely typed and for procedural programming, it provides flexibility to for more rigid implementations as well. This specific implementation of the system uses statically typed variables, especially in function arguments and return types. This is done to ensure that errors are thrown, and the program does not fail silently. Additionally, invalid values sent from the front-end should not fail silently and be accepted by the system.

The cryptographic libraries implemented in the system use an Object-Oriented Programming approach to clearly define object types for Keys, Key Parameters, and Encrypted Messages. All database tables and fields are also defined and represented as classes with properties and attributes, instances of which are saved as objects.

VanillaJS

Besides using plain HTML and CSS to display basic data through a web-based system in a browser, the only way to encrypt/decrypt votes and add functionality to a front-end page is to use a flavour of Javascript [100].

Javascript in itself has numerous options to choose from. Examples include JQuery, AngularJS, ReactJS, et al. The only caveat to making use of any of these flavours, despite them being open source, is that they get updated and upgraded regularly. Adding these files will either require importing from external links that are fetched on demand (which may include vulnerabilities in the system), or including a huge file at once (which is a haven for an adversary to inject vulnerabilities) [101].

Using Javascript in its basic (Vanilla) version ensures that the scripts used remain readable by most major browsers, have a longer lifecycle, and do not have external dependencies that need updates (or include yet unpatched vulnerabilities).

Javascript handles the ballot received from the server, including Election Metadata with unfilled Questions and Choices. The script manages how this information is presented to the users. It includes functions that record selections, encrypt choices, read uploaded credentials, package the final ballot once it has been marked and encrypted, and finally send the ballot to the bulletin board through a POST request.

Cryptographic Libraries

Since both Python and Javascript are used extensively, the community and developer support for both of these languages are active. Python offers a large number of libraries that help with cryptography, including PyCrypto, PyCryptodome, PyCharm, et al. Libraries like LibSodium (or NaCl) help with managing random and prime number in a manner favourable to cryptographic libraries.

On the client-side Javascript end, LibSodium is available for random numbers, while a BigInteger library becomes necessary to read and perform operations on large integer numbers that need encryption, re-encryption, and hashing.

7.4 Object Notations

ElGamal keys, key parameters, and encrypted messages need a standard representation in a language like Python that is flexible with variable types. A voting system like Resist also requires that these objects be represented in a standard manner across both, Python and Javascript, limiting change in form as it is transferred and read.

7.4.1 JSON

Javascript Object Notation (JSON) is a common way of representing objects that contain key-value pairs, with multiple value (variable types). Data is often represented as a JSON object when requests and responses are made over the internet.

JSON notations are easy to load as dictionaries in Python. The language has a simple, understandable approach towards reading from and creating JSON objects. The `json.dumps(⟨JSONobject⟩)` and `json.loads(⟨JSONobject⟩)` methods abstract these conversions for a Python developer.

Javascript usually reads JSON from strings sent in requests with minimal effort, and also converts these objects to a string using `JSON.strigify(⟨JSONobject⟩)` when there is a need to send an object (like a marked ballot) along with a POST request.

An example of an unmarked ballot received from the server is given below.

```
{
"electionMetadata": {
"electionUUID": "d7df4744-5571-45ce-a648-be97fb577be4",
"electionName": "President",
"electionOrganization": "Dalhousie",
"electionDescription": "Pres of Dal",
"electionPublicKey": {
"p": "13262045991911476461809003740714471490693696821777845865538537
```

```
19108935220067379668486583036830507351325720769602565836190683023378
5360833574615894770291807",
"q": "66310229959557382309045018703572357453468484108889229327692685
95544676100336898342432915184152536756628603848012829180953415116892
680416787307947385145903",
"g": "59800099574562864354455865079198977005954179037000538908195223
66719026409304366430660943870128216024867281863109362083887745515824
710411585446518267713671",
"y": "33103762896828498257572248228318329530412032654916271040201032
64732514728313487229684363292026096485646777495416082375930918846557
39736682368506849182537859443760937542977668952369416986233468794524
05284454223595840521128148219008070164511782848572382054800095482265
19517376543733050498758840510916212257015209747442004387005968093785
07849259723708906250793641115778099105116851913250986151839564009637
92635459039141484124683820682971604443950769976090497441180"
},
"electionHelpEmail": "supervisor@email.com",
"electionCastUrl": "/booth/castvote",
"encrypted_choices": [
{
"a": "1050422469831926424701835941597830042787109099914126564416
1881770487537846077314708818453072363188347770261598745785337566
738209125570415378780653777522230",
"b":
"824010627077503208720215365468048512884749344401244917214225252
7848659689770970765195013297149943630723442702719334589586356639
25157768432991853706992864"
},
{
"a": "1277551621331865810242556293507696041901125906823805570303
6881266387199424757860888442528964482087387686257658197390971084
448272799442150227207639624213953",
"b": "4164870262818918704650021232816313968065750484030981623837
4295454380758897855475742508267020926229861085811302505565235986
6040764937455112037025872525 8317"
}
],
"electionIndex": "/booth/",
"numberOfQuestions": 1
},
"questions": [{
"questionID": 6,
"questionText": "Who should be the president?",
"questionDescription": "Choose a president.",
"questionMaxChoices": 1,
"numberOfSelectedChoices": 0,
"choices": [
{
"id": 11,
"text": "Robbie",
"selection": 0
},
{
"id": 12,
"text": "MacG",
"selection": 0
}
]
}
]
}
```

7.4.2 Integers, BigIntegers, and Arrays

While integers in most languages have been restricted to 32-bits for short integers and 64-bit for long integers, Python unified the two in 2001, making the length of integers restricted only by available memory [102]. As a result, python allows for long random numbers used as cryptographic parameters to be treated as and operated upon as regular integers.

When transferring these values across to the client-side where Javascript has to read and perform operations on these values, the numbers appear to be too long to be treated as regular integers. To avoid rounding these up to an exponential notation, these numbers are sent as strings. Javascript does not alter or truncate strings and reads them in their original form.

As mentioned earlier, a library like BigInteger allows strings to be interpreted as long numbers that can be operated upon. BigInteger support in Javascript can be exploited to add, multiply, and mod these numbers, and eventually turn their encrypted outputs back into strings, all using Javascript. Previously, in the absence of BigInteger support, the browser's support for Java and JVM was relied upon. This was not only inconvenient, but it also posed a potential for introducing malicious code [52].

An example JavaScript function for reencrypting a given value looks as follows.

```
function elGamalReEncrypt(ciphertext, publicKey, params) {
r = bigInt.randBetween("1", params.p);
return {
a: bigInt(ciphertext.a)
.multiply(bigInt(params.g).modPow(r, params.p))
.mod(params.p),
b: bigInt(ciphertext.b)
.multiply(bigInt(publicKey).modPow(r, params.p))
.mod(params.p)
};
}
```

7.5 Modeling Theoretical Concepts

The concepts explained in Section 6.1 and Section 4.1, including representing objects, players, and entities in the proposed voting system are explained here. The explanation primarily focuses on the development of the database as an Object-Relational Model with reasoning provided towards certain design choices.

7.5.1 Users, Privileges, and Groups

Django provides its own Authentication system that can be used by default, substituted, and augmented. The system provides for creating and managing user accounts, authenticating user login, categorizing users into groups, and database interaction privileges based on user and group accounts. Most of Django's available user functionality can be imported from `django.contrib.auth` and the `AUTH_USER_MODEL`.

Django provides an `AbstractUser` class that can be extended to include additional fields required for users in the system. This is exploited to include an Authentication Key and Designation Key for each user. These keys need to be kept unaffected by current active and inactive elections and specific to each user that has, at least once, been authenticated to participate in elections. A visualization of this inheritance can be found in Figure 7.1.

User Groups are designed to represent the players in Resist. Permissions are also introduced for each group, so that any user is limited by the privileges of the group itself. Users are assigned to one5 of the following groups.

- **Supervisor.** The supervisor creates and manages the system and elections. They create election parameters, add questions (with choices), add registration tellers, tabulation tellers, and eligible voters to each election. They are responsible for initial key exchanges, generating an election's public key, and freezing registration and voting. They also initiate the tallying process and declare results.

- **Registration Teller.** The registration teller accounts created in Resist are made to mimic a groups of users with privileges. However, this exists only in the backend, with no support for templates associated with the app. Registration Tellers have private credentials (explained later) that are stored and accessed from an extended entry to a user model, called RegistrationTellerPrivate, as shown in Figure 7.2.

- **Tabulation Teller.** The tabulation teller accounts created in Resist are similar in structure and intention to the aforementioned Registration Teller group, barring that these include actual users that explicitly provide consent and private key values through a front-end template. Tabulation Tellers access a random shared value (z), along with their share of the private key, that they store and access from an extended entry to a user model, called TabulationTellerPrivate, as shown in Figure 7.2.

- **Voter.** Every user is considered a voter in the system, regardless of their additional role for specific elections. Voters have the ability to use their authentication and designation key to register for an election. If registered, they have access to a specific credential that is made available to them (for one time only), and a corresponding entry is added for them in the RegisteredVoter model (which is explained later).

7.5.2 Implementing the Bulletin Board

In order to ensure that the Bulletin Board is kept independent of the players in the system, the Bulletin Board is sequestered into its own Django App, `bulletin`. The app contains a model file that defines elections and the corresponding attributes necessary to be kept accessible not just to users registered in the system, but also to any access requests made without the need to login.

Each `Election` has a specific user, the `supervisor`, that creates the election. The `Election` is identified by a simple UUID. Once the supervisor has defined the parameters and created the election, they can proceed to manage it. The tabulation tellers can receive private key shares, and the election will, as a result, receive a single public key. When the keys have been generated, possible choices are encrypted using the public key, and added to the model.

7.5.3 Supervisor

The supervisor is a user with `staff` privileges. This implies that they can interact with Django's admin panel to create and manage election parameters. Once logged in, a supervisor is presented with the option to either create an election or manage an existing one. Choosing to create a new election takes them to the admin panel. This transition can be viewed in Figure 7.4.

Once the supervisor has created and defined an election, they can use the Admin Panel to add existing users to the `Eligible Voter`, `Authenticated Teller`, `Registration Teller`, or `Tabulation Teller` models. Note that key exchanges are not performed here, since it's still considered as a part of 'creating' an election.

After adding tellers and participants to an election, the supervisor can return to the index page to Manage an Election. Managing an Election includes performing key exchanges, freezing registration and voting, and releasing the results of an election. Each of these scenarios can be found listed in the subfigures of Figure 7.5.

7.5.4 Voter Lists

Resist makes an attempt to introduce verifiability of the participants in an election. The Eligible Voters in the system comprise all potential voters for elections conducted across the system. These voters do have identifiable information associated with them and access to a single user account per entry. The list is made publicly accessible before the start of any election so that any illegitimate entry can be pointed out and rectified.

When a voter wishes to exercise their right and participate in an election, they can employ an untappable channel (show up physically) to identify themselves and obtain an authentication key and a designation key. While Resist does not have an implementation set up to handle the transfer of these keys, it currently adds the private key to the user's account on the system and adds the public key to their corresponding entry on the Authenticated Voter list. This list is created as a subset of the Eligible Voter list to prevent any other account from participating in upcoming elections.

When an election has been created and the keys have been generated, voters in the Authenticated Voter list can register to obtain their credentials and participate in the election. Once they receive their credentials (using their keys), they are added as an entry on the registered voter list. This list, as before, is a subset of the Authenticated Voter List. This makes it easier to ensure that entries weren't made by colluding registration tellers.

While it was possible to determine illegitimate entries from the number of credential shares, constructing the model as shown in Figure makes it difficult for adversaries to surpass the structure of the model.

7.5.5 Registration Tellers

Resist implements Registration Tellers within a separate *register* django app. Although there is no explicit user interface developed for the tellers, they have their own user accounts that fall under the 'Registration Teller' group. These accounts have a related table Registration Teller Private that stores their private key. This was shown in Figure 7.2.

When a registers for an election, they choose to do so from the front-end (index page) for the election that takes them to a list of elections available for registration. This is shown in Figure 7.8.

When the user chooses the election, the server-side script for the *booth* app sends a request to each registration teller. The request is handles by the *register* app. The two apps perform the Needham-Schroeder-Lowe protocol to authenticate each other and obtain credential shares.

The Credential and Credential Share models are updated. They are associated with the appropriate Registered Voter model in order to associate it with a public index. The Registered Voter model serves as a parent so that entries for a credential share is associated with a specific teller alone. This setup can be inferred from Figure 7.9.

Once the protocol has been jointly executed, the 'booth' server-side for the user can decrypt their credential shares and verify it against the values published on the bulletin board.

Resist assumes one honest registration teller and alters the share obtained from them. A Fake DVRP is generated for this new credential. In order to demonstrate this mechanism, Resist presents multiple fake credential options to the user, each corresponding to a different honest teller. It is recommended that a separate interface be provided to the user to help them generate their own fake credentials.

The credentials are made available to download as JSON files. This is shown in Figure 7.10. An implementation of this system needs to present an alternative, one that remains an open question.

7.5.6 Casting a Vote

When a voter wishes to participate in an election, they need to ensure that they are not logged into the system. Resist does not track activity linking it to a specific account. This is done to ensure that anyone can anonymously cast a vote without letting the system or an adversary know that they are registered to participate.

To audit a ballot or cast a vote, the voter can navigate to do do so from the system's index page. They will only be shown a list of elections that are active and have their registration frozen by the supervisor. This is shown in Figure 7.11.

Once they choose the election that they wish to participate in, the entire ballot gets downloaded. They can choose to disconnect their connection to a network or the internet and mark their ballot locally. All relevant javascript files required for the generation of proofs and encryption are downloaded when the page loads. This is done to ensure that an eavesdropper does not witness plaintext ballots that have been marked.

Voters are initially presented with the metadata for each election. They can use this verify the correctness of the ballot against the bulletin board. Information regarding support (help email) is also included. This can be seen in Figure 7.12.

The user is presented one question at a time, with the choices listed under it. All details regarding the question, including a description of the question and maximum number of choices, are displayed on the same page. Users can navigate to the next question and back, or choose to review their choices so far. The Javascript code associated with the the ballot restricts the number of choices that the user can select. Figure 7.13 shows a question as presented at the booth.

Once the questions have been marked, the voter can review their ballot in plaintext before proceeding to encrypt them. The plaintext marked choices are discarded along with the original ballot with its metadata when the choices are encrypted. The review as presented to the voter is shown in Figure 7.14.

Figure 7.15 displays the encrypted choices to the user. At this point, all records of the plaintext ballot are deleted and only relvant fields are retained.

Encrypting choices in the front-end is merely a re-encryption of one of the elements from the array of encrypted_choices sent along with the ballot.

The voter can send this to the bulletin board for audit. Encrypted ballots for audit have to be posted without the credential in order to maintain coercion resistance. If they intend to cast this ballot instead, they can proceed to upload their credentials.

The credential JSON file received during registration can be uploaded from the local disk. Once imported, the interface allows for them to review the contents of the credential before adding them to the ballot, as shown in Figure 7.16.

Once the choices have been encrypted and the credential has been uploaded, the voter gets one final review of the overall ballot. They get to view the entire ballot that is to be cast. This also serves as a receipt for the voter that they can use to verify against the bulletin board. Figure 7.17 shows the final form of the ballot. The voter can then connect back to the network or the Internet and cast their ballot to the bulletin board.

7.5.7 Tabulation Tellers

Once votes have been cast, the supervisor can freeze voting so that no new votes are cast. Since Resist has been implemented as a Distributed Cryptosystem, all tabulation tellers are required in order to compute the tally.

The tally is initiated by the supervisor of the election, but is conducted overall by tabulation tellers. The tellers interact with each other through to compute decryptions and verify proofs. The setup of the model is given on the bulletin board as shown in Figure 7.18.

This phase is computed according to the algorithm shown in the previous chapter. Each mix server is a part of the tabulation app, corresponding to each teller.

The results are then made available on the bulletin board and the election is declared inactive by the supervisor. All encryptions, proofs, and results remain on the database (on the bulletin board) to encourage future audits of the system.

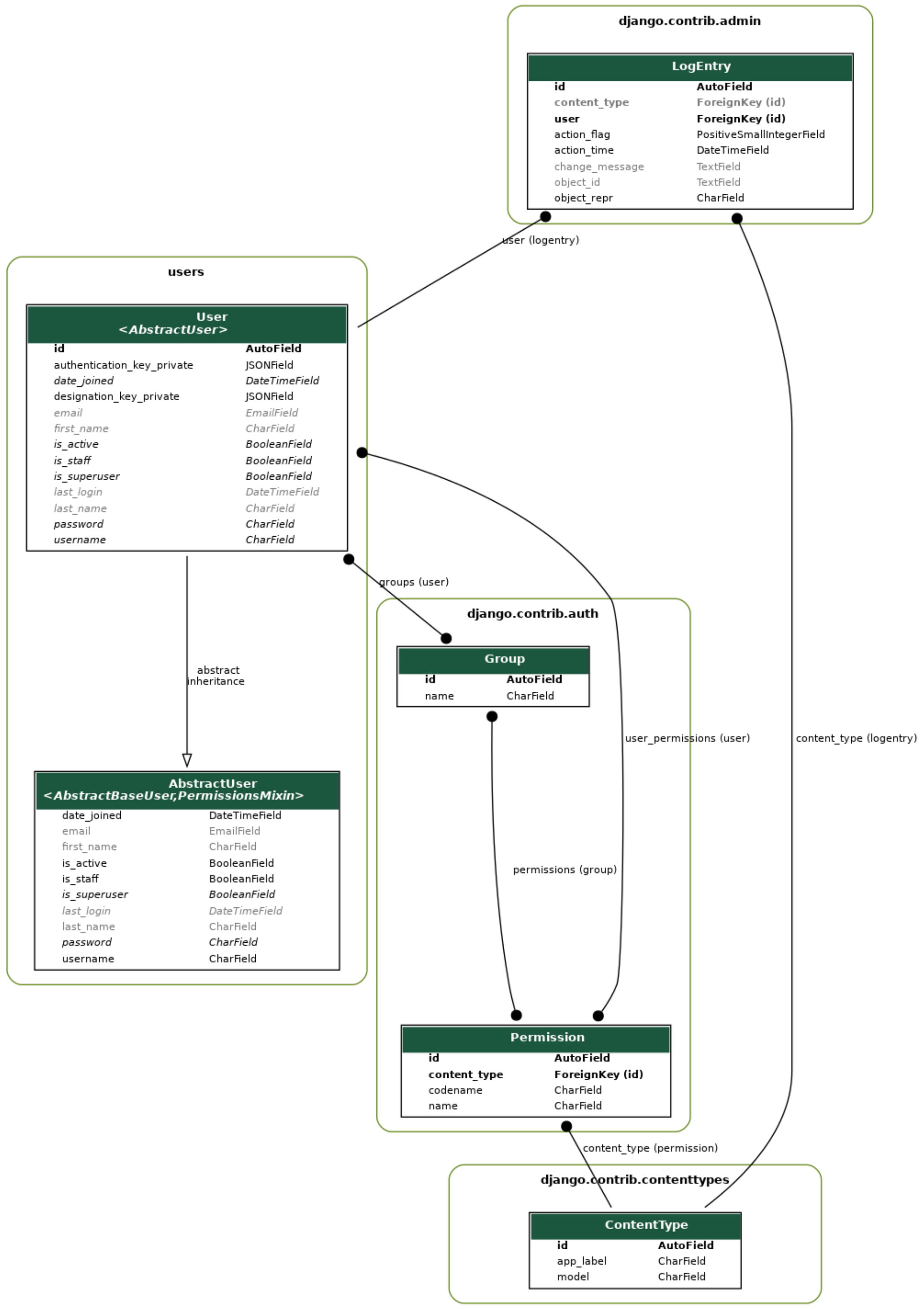
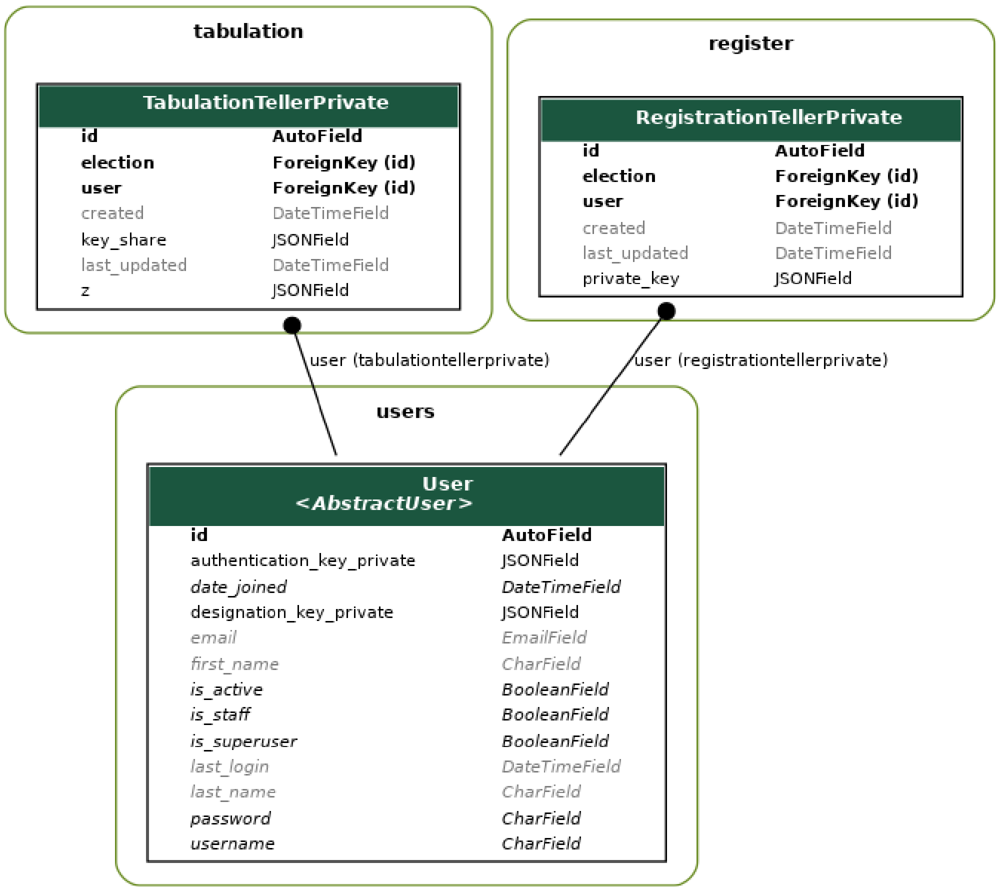


Figure 7.1: Django's default user model has been expanded to include the Authentication Key and Designation Key associated with each Authenticated Voter in Resist. The ContentType, Permission, Group, AbstractUser fields are created and managed by Django, and are inherited by the custom 'User' model. LogEntry is created and updated to maintain a record of user interaction.


tabulation
register
user (tabulationtellerprivate)
user (registrationtellerprivate)
users

TabulationTellerPrivate	
id	**AutoField**
election	**ForeignKey (id)**
user	**ForeignKey (id)**
created	DateTimeField
key_share	JSONField
last_updated	DateTimeField
z	JSONField

RegistrationTellerPrivate	
id	**AutoField**
election	**ForeignKey (id)**
user	**ForeignKey (id)**
created	DateTimeField
last_updated	DateTimeField
private_key	JSONField

User *<AbstractUser>*	
id	**AutoField**
authentication_key_private	JSONField
date_joined	*DateTimeField*
designation_key_private	JSONField
email	*EmailField*
first_name	*CharField*
is_active	*BooleanField*
is_staff	*BooleanField*
is_superuser	*BooleanField*
last_login	*DateTimeField*
last_name	*CharField*
password	*CharField*
username	*CharField*

Figure 7.2: The User Model is further customized to include fields required by specific types of users. These entries are only created at the time of creating and managing elections. Private Values remain accessible only to users that have been included in the corresponding group.

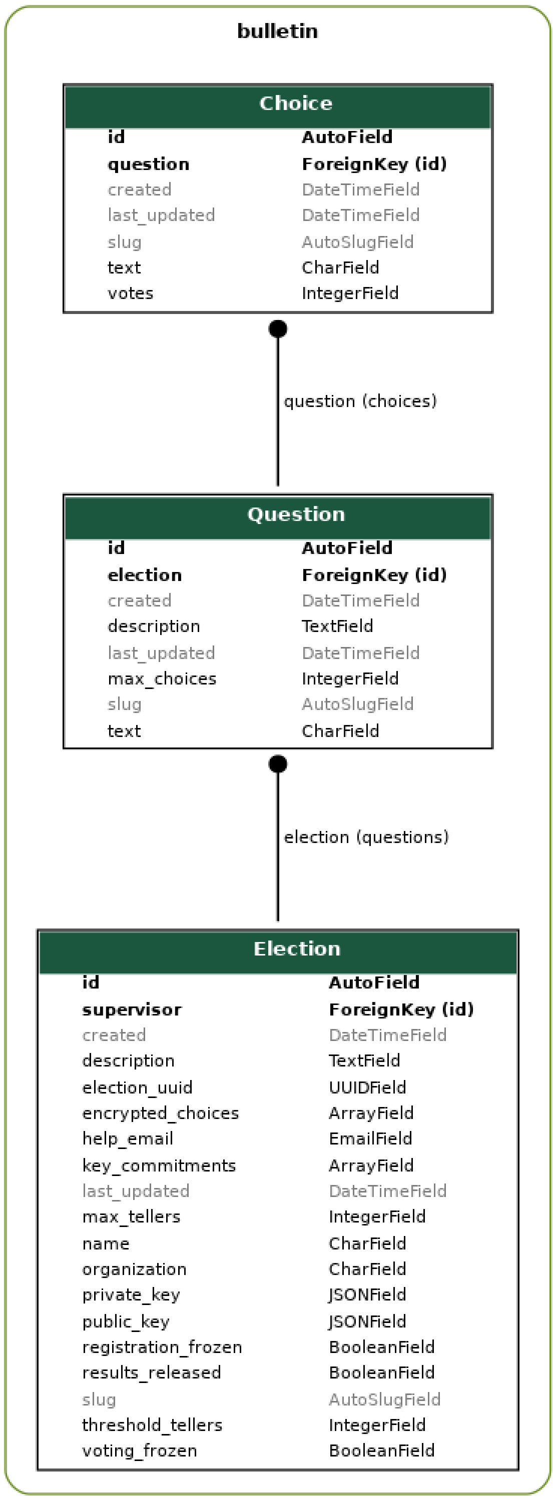
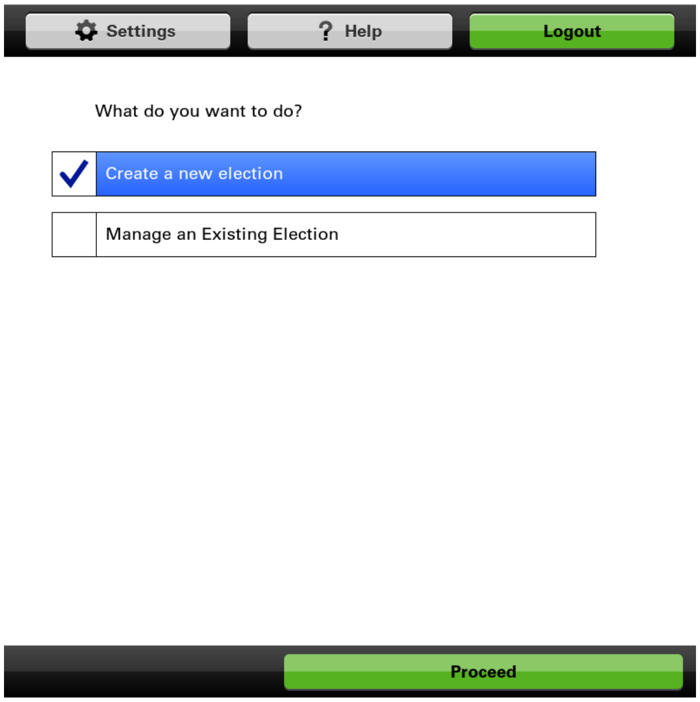
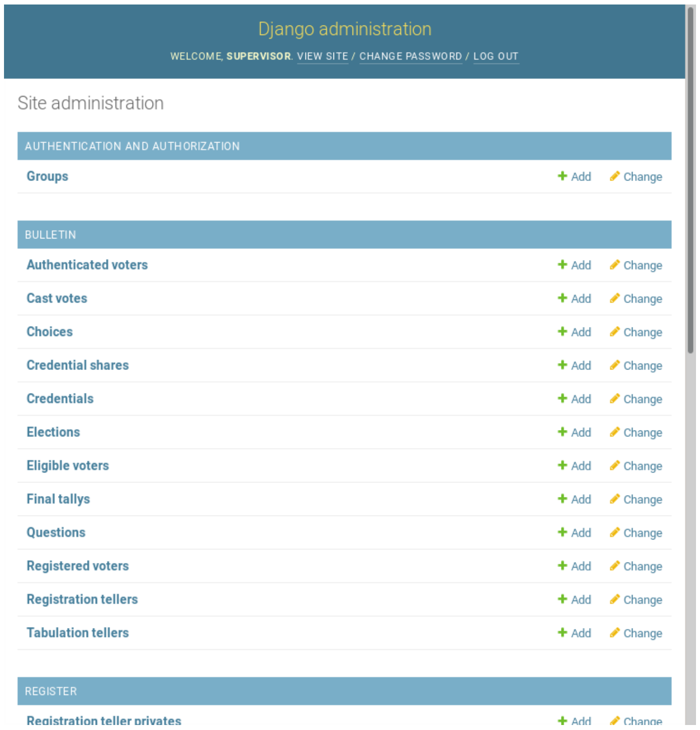


Figure 7.3: The `Election` Model includes the requisite metadata necessary to define and conduct an election. The `Question` and `Choice` Models are kept as inheritances to provide them with unique identifiers and pre-defined fields.



(a) Supervisor Front-End


Django administration

WELCOME, SUPERVISOR. VIEW SITE / CHANGE PASSWORD / LOG OUT

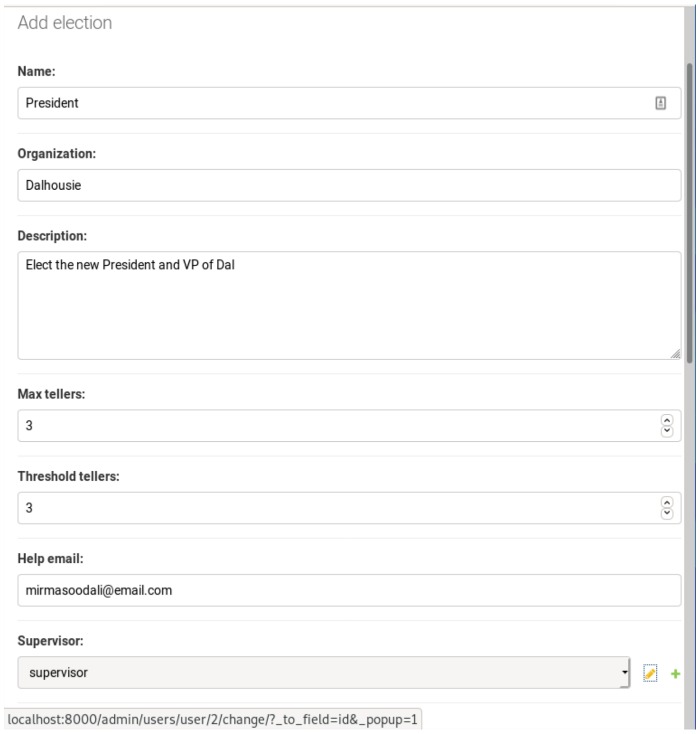
Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change

BULLETIN		
Authenticated voters	+ Add	Change
Cast votes	+ Add	Change
Choices	+ Add	Change
Credential shares	+ Add	Change
Credentials	+ Add	Change
Elections	+ Add	Change
Eligible voters	+ Add	Change
Final tallys	+ Add	Change
Questions	+ Add	Change
Registered voters	+ Add	Change
Registration tellers	+ Add	Change
Tabulation tellers	+ Add	Change

REGISTER		
Registration teller privates	+ Add	Change

(b) Admin Panel



Add election

Name: President

Organization: Dalhousie

Description: Elect the new President and VP of Dal

Max tellers: 3

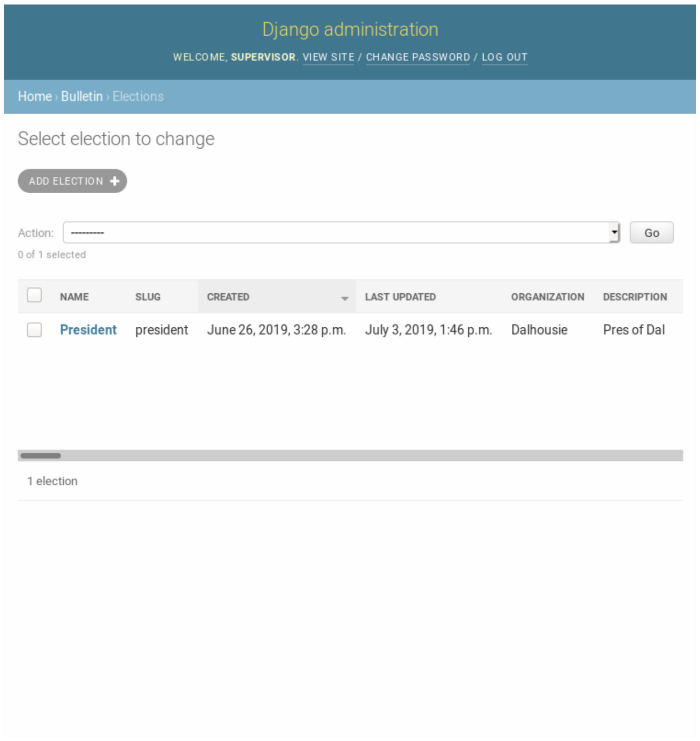
Threshold tellers: 3

Help email: mirmasoodali@email.com

Supervisor: supervisor

localhost:8000/admin/users/user/2/change/?_to_field=id&_popup=1

(c) Admin Form for creating an `Election`



Django administration

WELCOME, SUPERVISOR. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Bulletin › Elections

Select election to change

ADD ELECTION

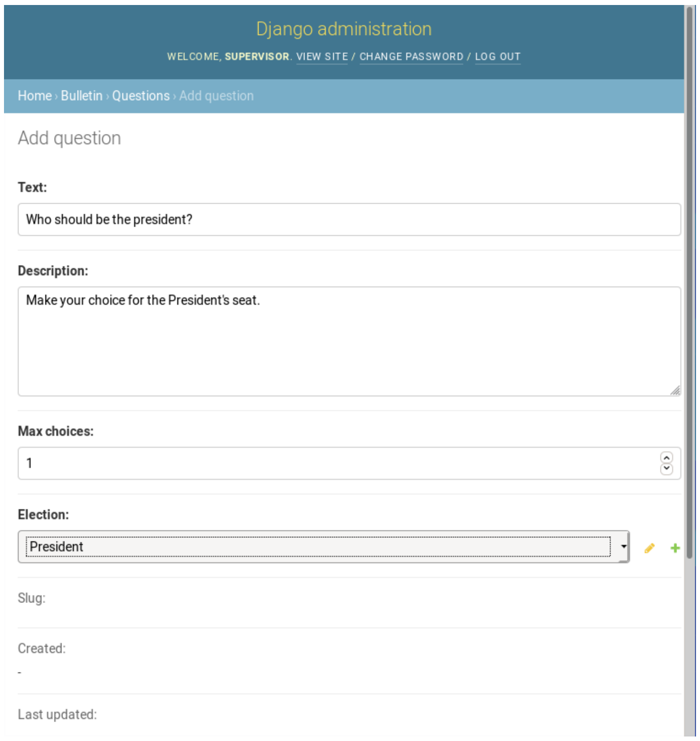
Action: --------- Go

0 of 1 selected

NAME	SLUG	CREATED	LAST UPDATED	ORGANIZATION	DESCRIPTION
President	president	June 26, 2019, 3:28 p.m.	July 3, 2019, 1:46 p.m.	Dalhousie	Pres of Dal

1 election

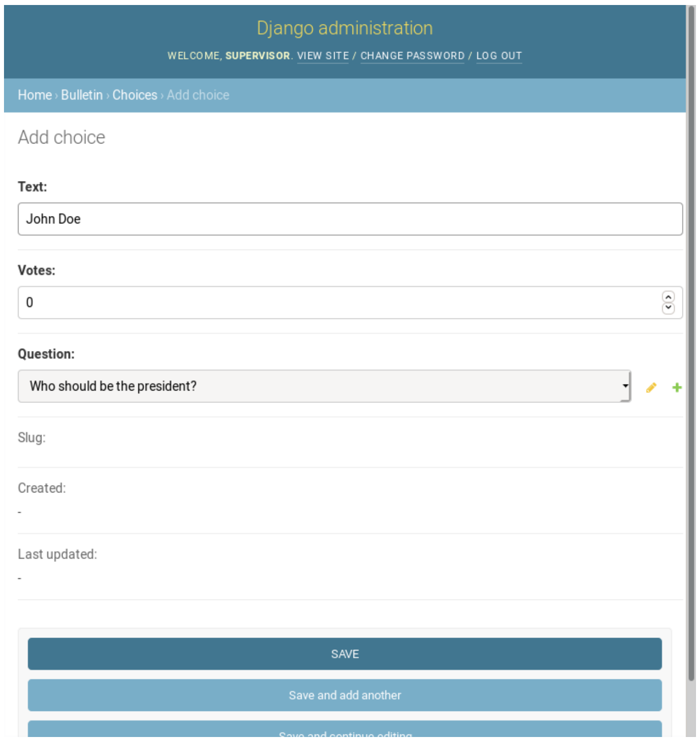
(d) Once the new `Election` has been created, it can be accessed from the same panel.


Django administration
WELCOME, SUPERVISOR. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Bulletin › Questions › Add question

Add question

Text:	Who should be the president?
Description:	Make your choice for the President's seat.
Max choices:	1
Election:	President
Slug:	
Created:	-
Last updated:	

(e) Admin Form for creating a `Question`

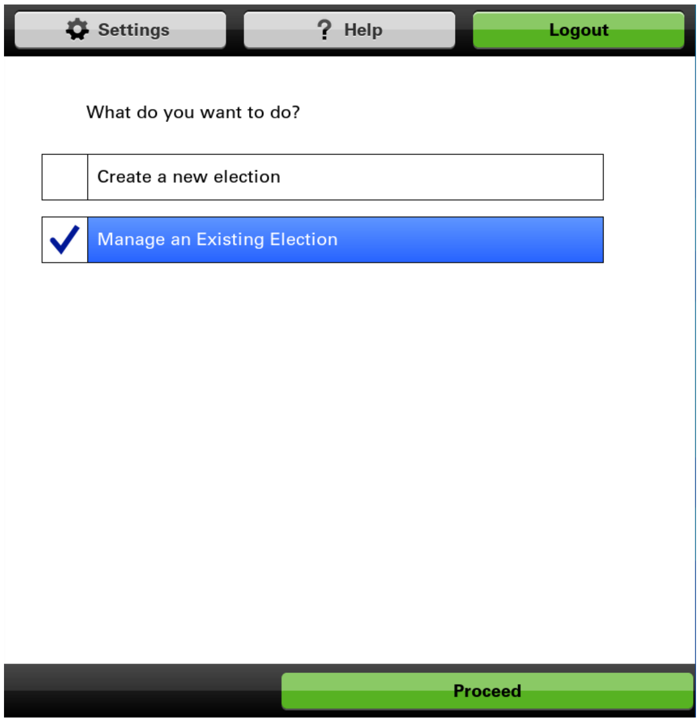

Django administration
WELCOME, SUPERVISOR. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Bulletin › Choices › Add choice
SAVE
Save and add another

Add choice

Text:	John Doe
Votes:	0
Question:	Who should be the president?
Slug:	
Created:	-
Last updated:	-

(f) Admin Form for creating a `Choice`

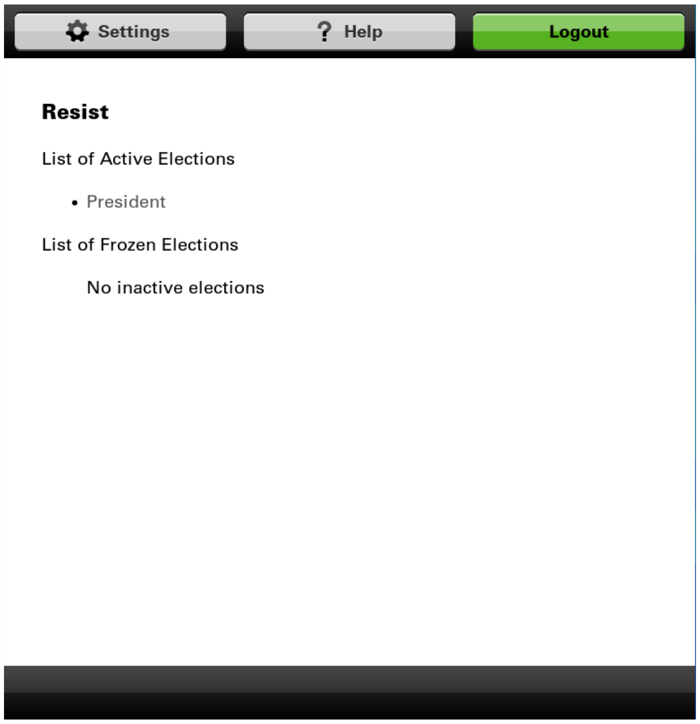
Figure 7.4: The index page presented to the supervisor offers options to Create a new election or to Manage an existing election. If they choose to create a new election, they are navigated to Django's Admin Panel which gives them access to the Bulletin Board.



What do you want to do?

- [ ] Create a new election
- [✓] Manage an Existing Election

(a) Supervisor Front-End, which they can use to manage an existing Election



Resist

List of Active Elections

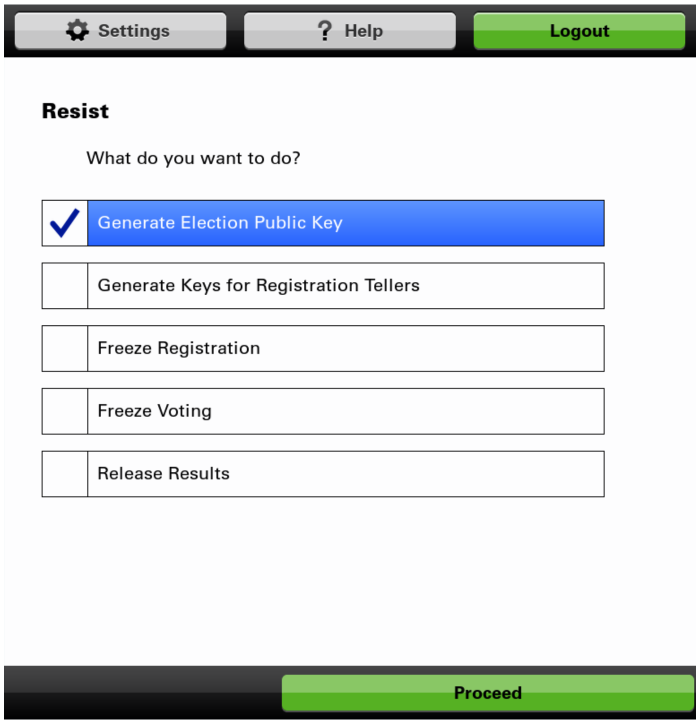
- President

List of Frozen Elections

No inactive elections

(b) They are then presented a list of Elections for which they are the supervisor

Figure 7.5: Managing Existing Elections

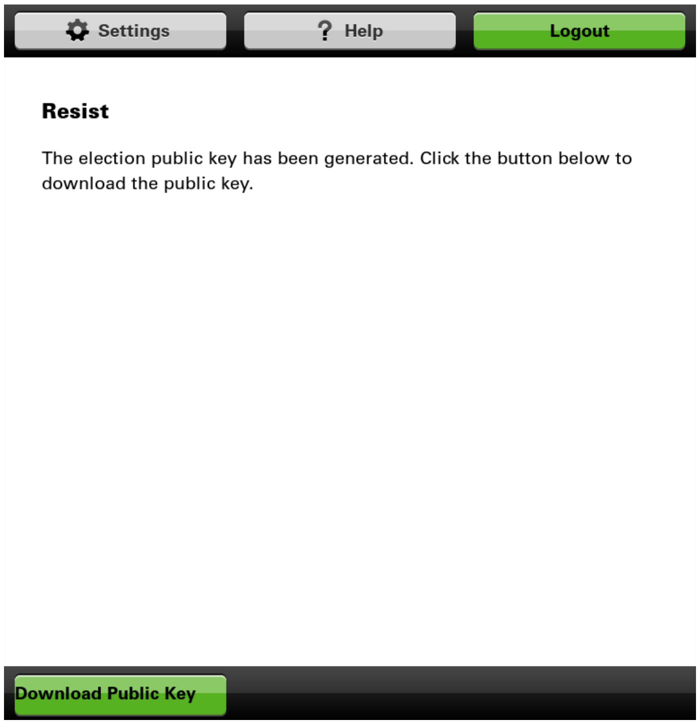


Resist

What do you want to do?

- [✓] Generate Election Public Key
- Generate Keys for Registration Tellers
- Freeze Registration
- Freeze Voting
- Release Results

(a) Supervisor Front-End, from which they can choose to generate the election's public key



Resist

The election public key has been generated. Click the button below to download the public key.

(b) As a result of selecting this operation, a public key is generated for the election. Tabulation tellers additionally receive their own share of the private keys.

Figure 7.6: Creating an Election's Public Key

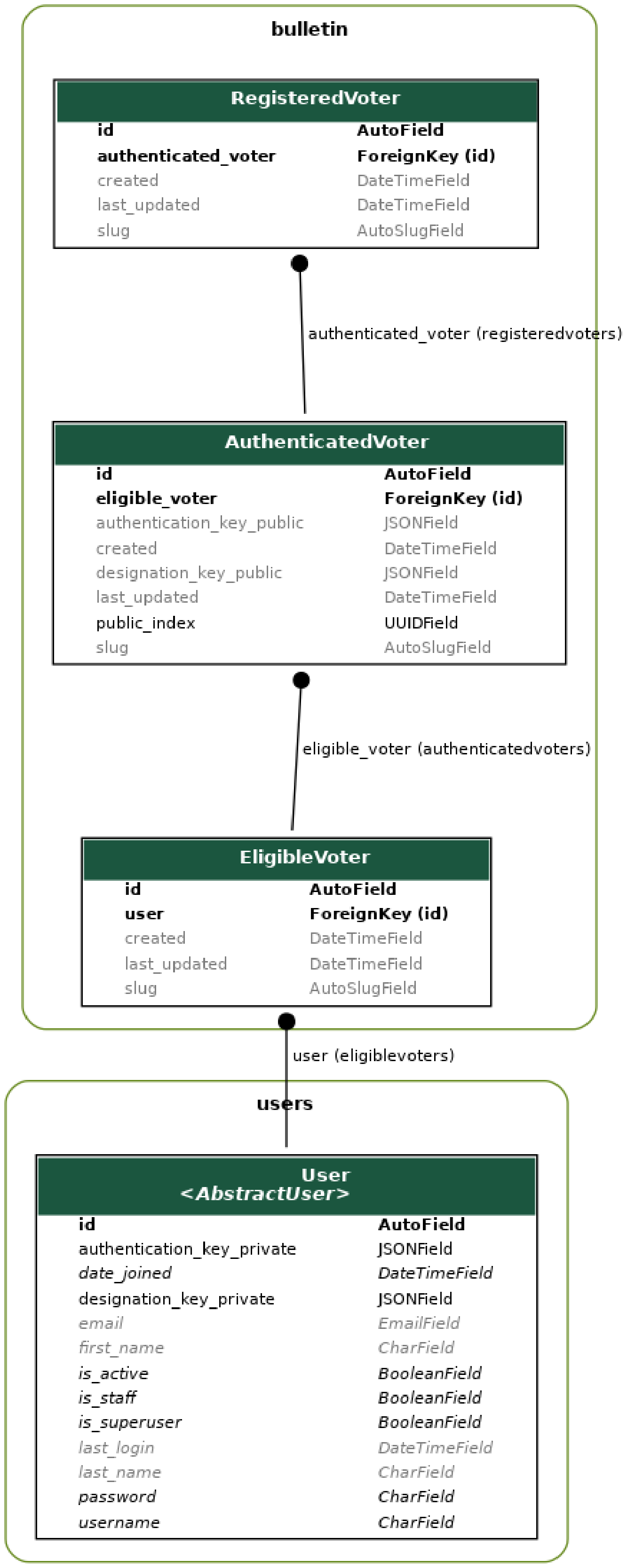
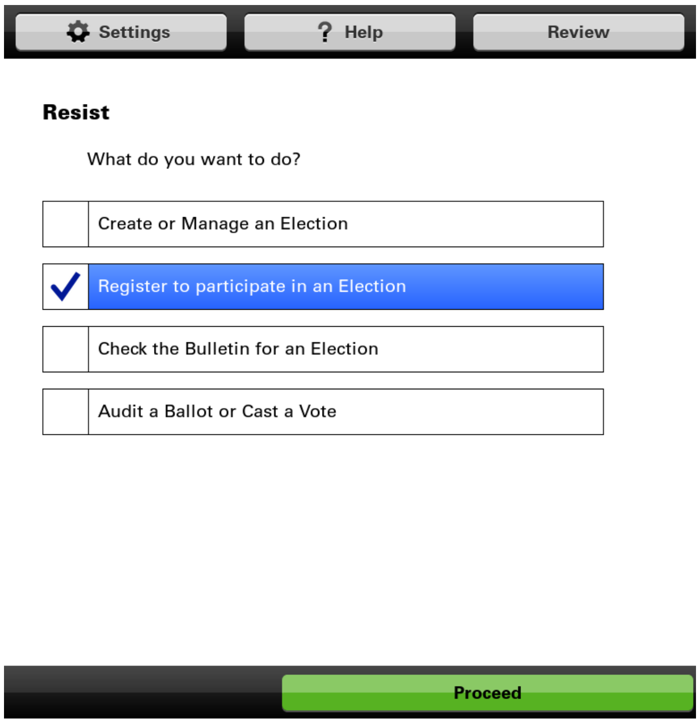


Figure 7.7: Django's model and inheritence of the three voter lists, created primarily to ease the verifiability of the Resist setup

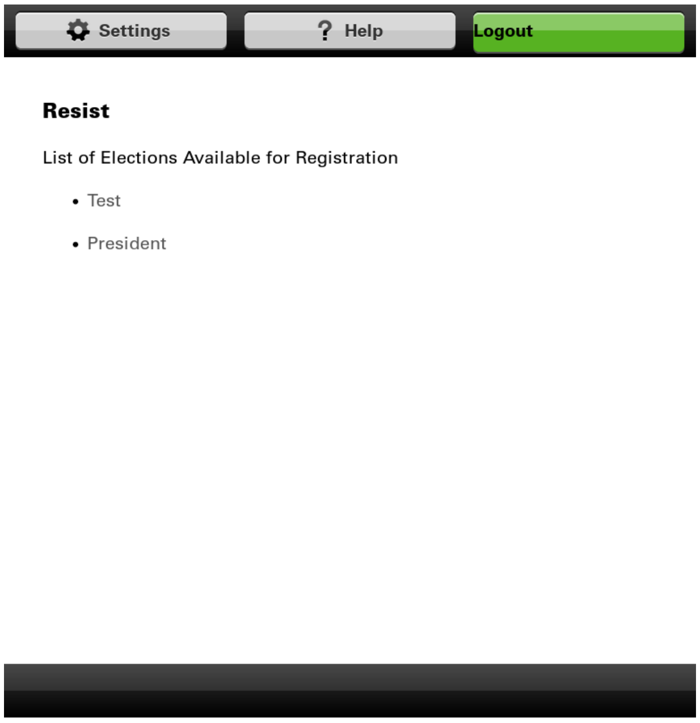


Resist

What do you want to do?

- [ ] Create or Manage an Election
- [✓] Register to participate in an Election
- [ ] Check the Bulletin for an Election
- [ ] Audit a Ballot or Cast a Vote

(a) The voter can choose to register for an election from the index page of the system



Resist

List of Elections Available for Registration

- Test
- President

(b) On a successful login, they are presented with a list of active elections for which they are eligible to register. The system also checks if registration for the election is yet to be frozen.

Figure 7.8: Registering for an election

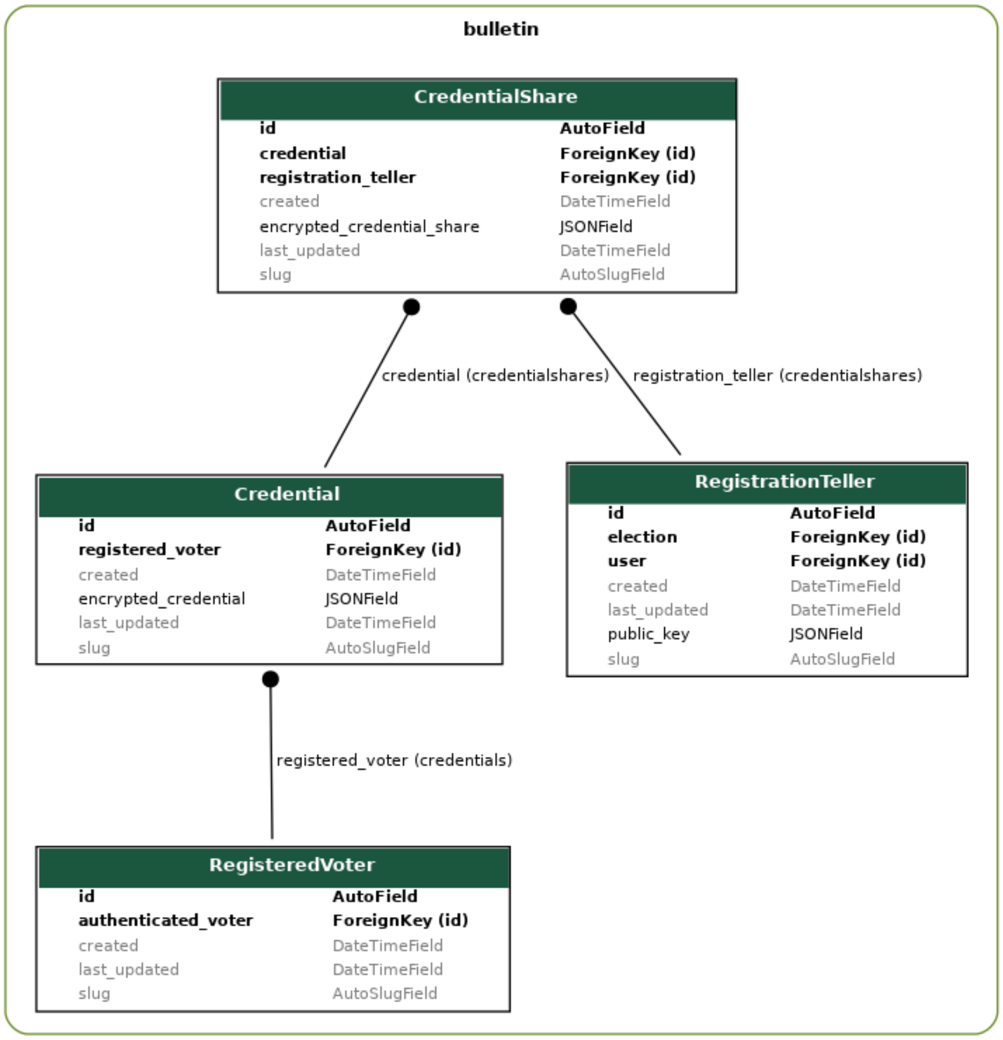
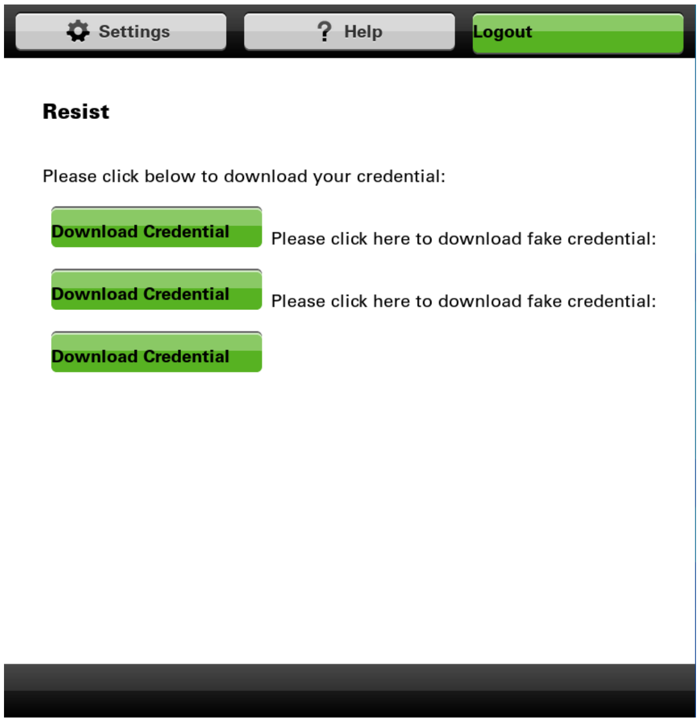

bulletin
CredentialShare
id AutoField
credential ForeignKey (id)
registration_teller ForeignKey (id)
created DateTimeField
encrypted_credential_share JSONField
last_updated DateTimeField
slug AutoSlugField
credential (credentialshares)
registration_teller (credentialshares)
Credential
id AutoField
registered_voter ForeignKey (id)
created DateTimeField
encrypted_credential JSONField
last_updated DateTimeField
slug AutoSlugField
RegistrationTeller
id AutoField
election ForeignKey (id)
user ForeignKey (id)
created DateTimeField
last_updated DateTimeField
public_key JSONField
slug AutoSlugField
registered_voter (credentials)
RegisteredVoter
id AutoField
authenticated_voter ForeignKey (id)
created DateTimeField
last_updated DateTimeField
slug AutoSlugField

Figure 7.9: Django's model and inheritence for posting Encrypted Credential Shares on the Bulletin Board.



Resist

Please click below to download your credential:

Download Credential

Please click here to download fake credential:

Download Credential

Please click here to download fake credential:

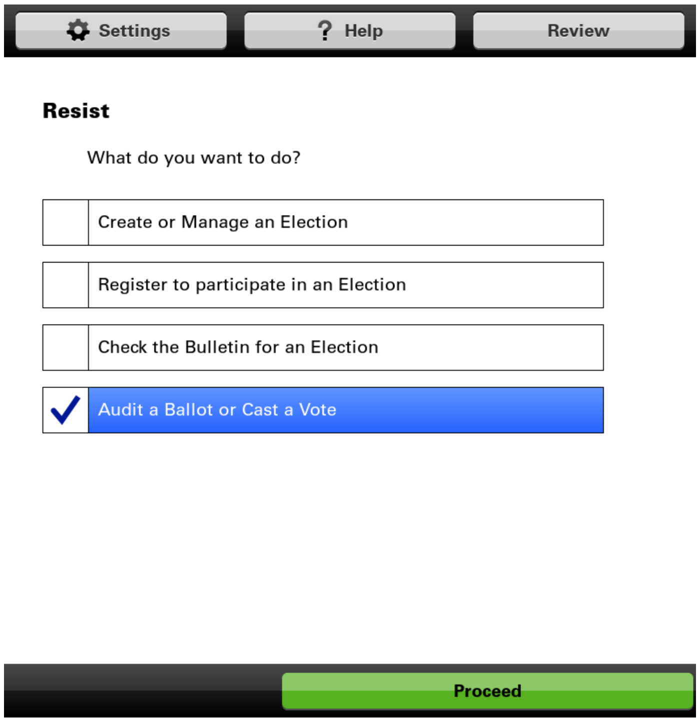
Download Credential

(a) The voter is presented (one-time-only) with the option to download their credential and credential shares as a JSON file.

```
{"credential_shares":["559140538342377773079599265390231591830136618009800580499275204252085887147829341500114473799273793861510181180046315861019490159105419078891372252777 1238",
"94404788607325470978757890441018833862638168489256552006852362723705834215621021453339392148953694033864091817264532673941539894209323460020605527414 11088"],
"credential":"710871554099174931328913379511272342823557674673057764401977973776373118061756641929412210026045374904270731781855891998582411320881847814378262786308978 4",
"encrypted_credential":
{"a":"3689910373190287426199404176775275414610750555742753212049631304225347739648954168375039834825911295820487089871965327542562400208739205811296267669807823",
"b":"74442891278308122935043812517194334092028377264383208099811584763381151906008332518955743399671604285005504286873040117511717049701235671847825901671292 80"},
"public_index":{"a":"98768936612141823672979389668149271870911425080171687482518695359753328796068852353463544915790334138276485390911483465717618643758648997754241189829 58687",
"b":"123780695911393483472248774785961908470870730591551964003250345032039696086769406365543952664651270095278945955957940638191433005555688851943178734764695 2"}}
```

(b) An example of what the JSON file can contain.

Figure 7.10: Obtaining Credentials and Credential Shares for an Election



Resist

What do you want to do?

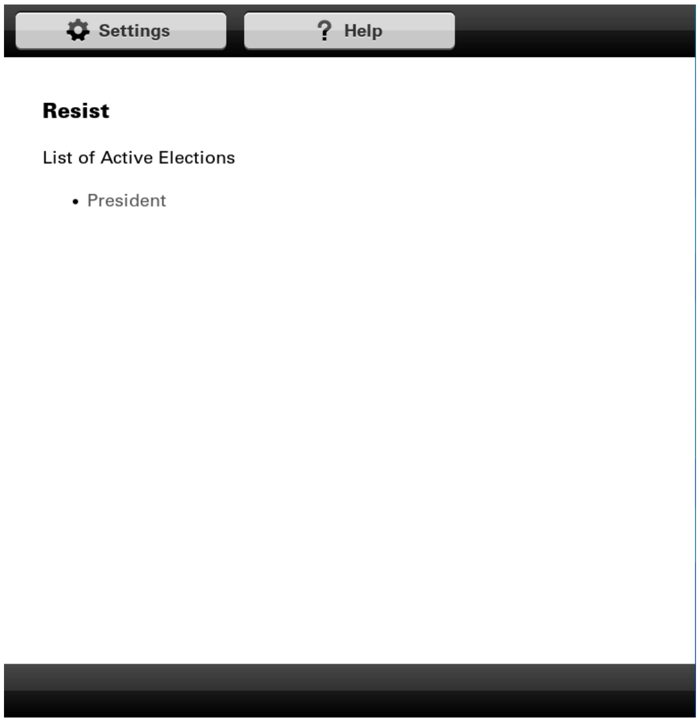
Create or Manage an Election

Register to participate in an Election

Check the Bulletin for an Election

✓ Audit a Ballot or Cast a Vote

(a) The voter can choose to audit a ballot or cast a vote. They are sent to the same booth regardless of their choice.



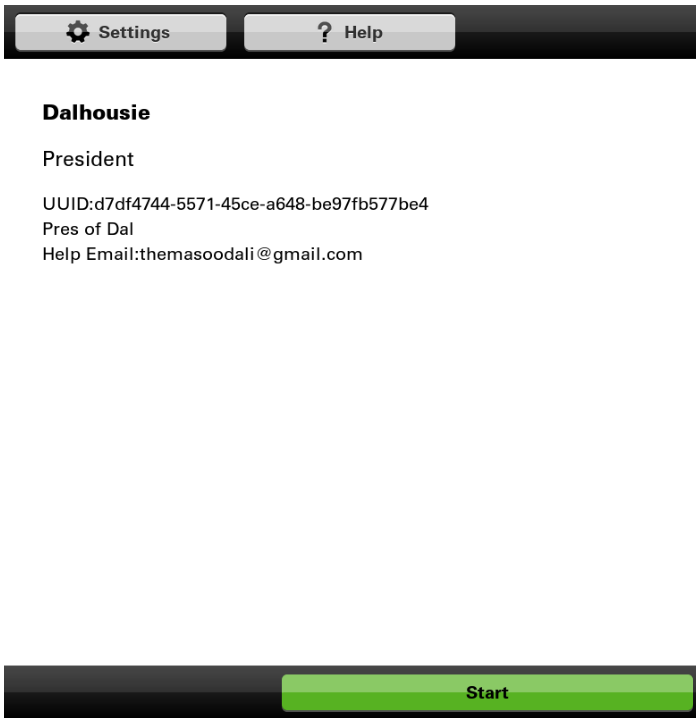
Resist

List of Active Elections

- President

(b) They can then choose from a list of active elections whose registration has been frozen.

Figure 7.11: Choosing an Election Ballot to Audit or Mark

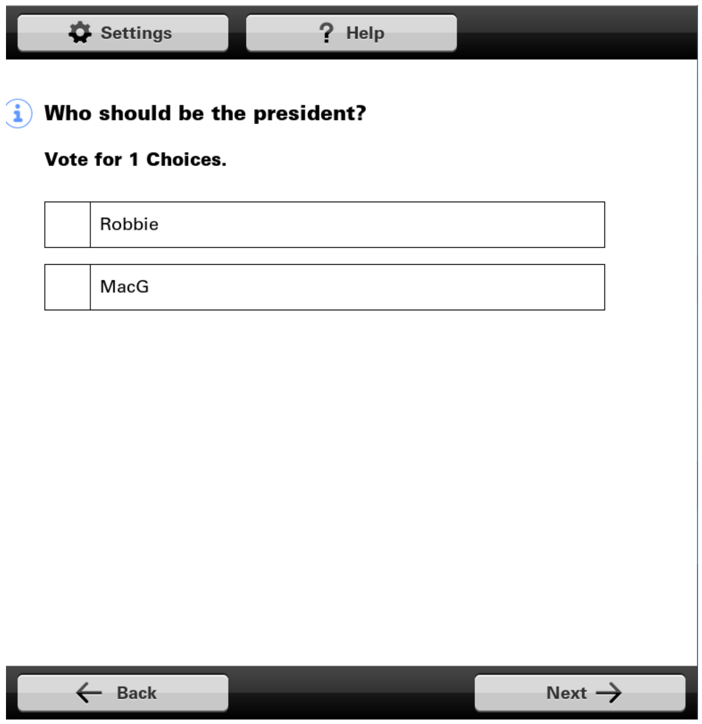


Dalhousie

President

UUID:d7df4744-5571-45ce-a648-be97fb577be4
Pres of Dal
Help Email:themasoodali@gmail.com

Figure 7.12: Details of the Election are displayed for verification



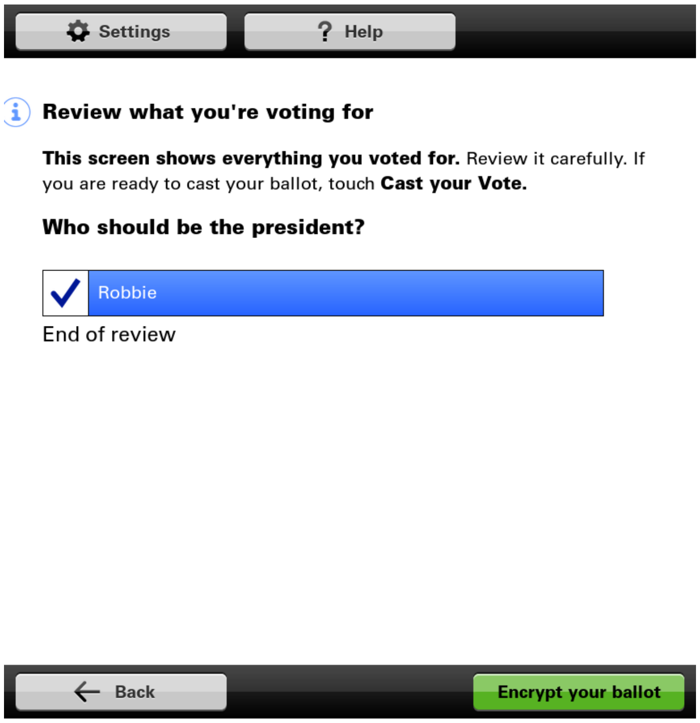
Who should be the president?

Vote for 1 Choices.

Robbie

MacG

Figure 7.13: One question is presented to a user at a time.



Review what you're voting for

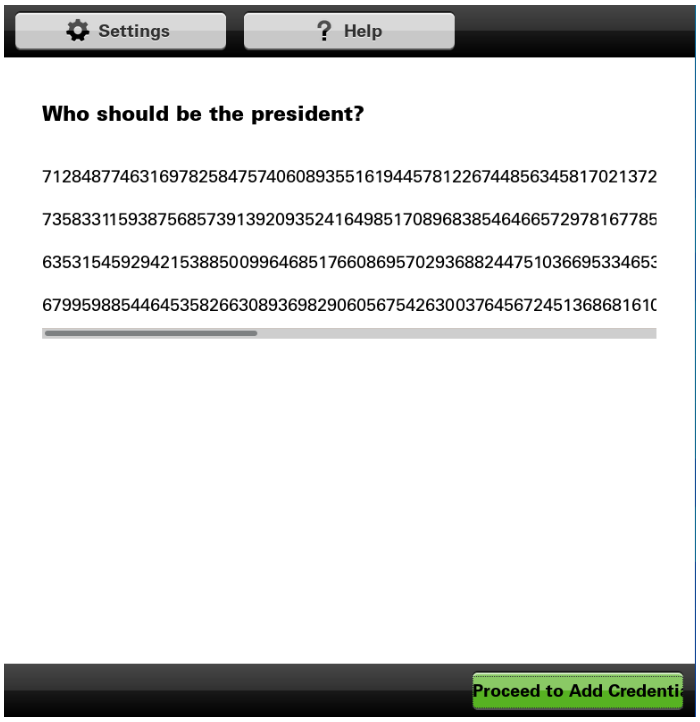
This screen shows everything you voted for. Review it carefully. If you are ready to cast your ballot, touch **Cast your Vote.**

Who should be the president?

✓ Robbie

End of review

Figure 7.14: The marked ballot is displayed for review before encryption.



Who should be the president?

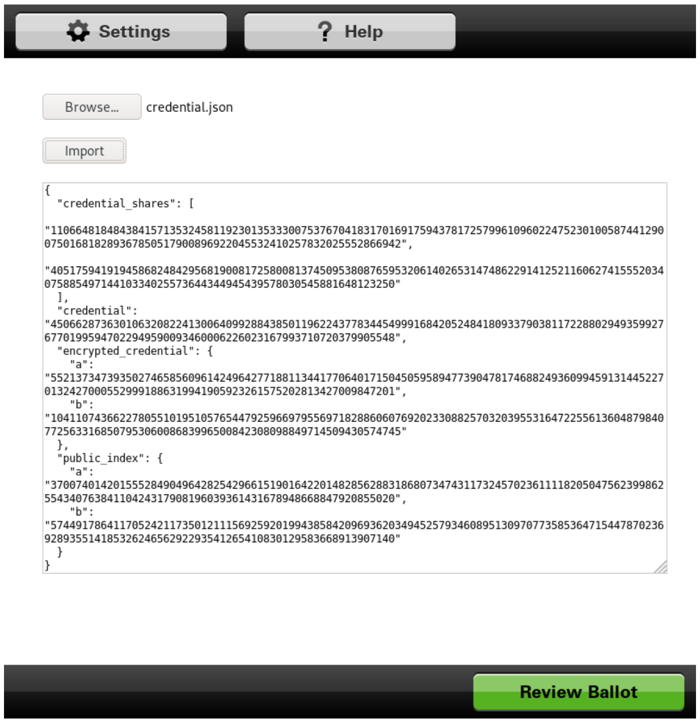
7128487746316978258475740608935516194457812267448563458170213372

73583311593875685739139209352416498517089683854646657297816778 5

6353154592942153885009964685176608695702936882447510366953346533

679959885446453582663089369829060567542630037645672451368681610

Figure 7.15: The marked ballot is encrypted and the resulting choices are shown to the user. This can be sent to the bulletin board to audit the encryption without the credential.

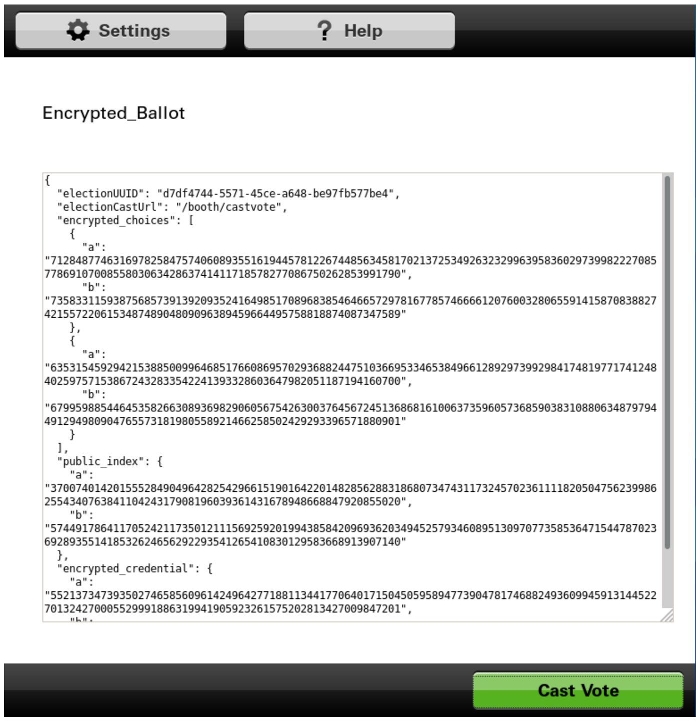


```
{
  "credential_shares": [

"11066481848438415713532458119230135333007537670418317016917594378172579961096022475230100587441290
07501681828936785051790089692204553241025783202555286694 2",

"40517594191945868248429568190081725800813745095380876595320614026531474862291412521160627415552034
0758854971441033402557364434494543957803054588164 8123250"
  ],
  "credential":
"45066287363010632082241300640992884385011962243778344549991684205248418093379038117228802949359927
6770199594702294959009346000622602316799371072037990 5548",
  "encrypted_credential": {
    "a":
"55213734739350274658560961424964277188113441770640171504505958947739047817468824936099459131445227
01324270005529991886319941905923261575202813427009847201",
    "b":
"10411074366227805510195105765447925966979556971828860607692023308825703203955316472255613604879840
7725633168507953060086839965008423080988497145094305747 45"
  },
  "public_index": {
    "a":
"37007401420155528490496428254296615190164220148285628831868073474311732457023611118205047562399862
55434076384110424317908196039361431678948668847920855020",
    "b":
"57449178641170524211735012111569259201994385842096936203494525793460895130970773585364715447870236
92893551418532624656292293541265410830129583668913907140"
  }
}
```

Figure 7.16: The JSON file for a credential can be uploaded from local storage. On opting to import the file, the contents are displayed for the user to review.



```
{
  "electionUUID": "d7df4744-5571-45ce-a648-be97fb577be4",
  "electionCastUrl": "/booth/castvote",
  "encrypted_choices": [
    {
      "a":
"71284877463169782584757406089355161944578122674485634581702137253492632329963958360297399822270 85
7786910700855803063428637414117185782770867502628539917 90",
      "b":
"73583311593875685739139209352416498517089683854646657297816778574666612076003280655914158708388 27
4215572206153487489048090963894596644957588188740873475 89"
    },
    {
      "a":
"63531545929421538850099646851766086957029368824475103669533465384966128929739929841748197717412 48
4025975715386724328335422413933286036479820511871941607 00",
      "b":
"67995988544645358266308936982906056754263003764567245136868161006373596057368590383108806348797 94
4912949809047655731819805589214662585024292933965718809 01"
    }
  ],
  "public_index": {
    "a":
"37007401420155528490496428254296615190164220148285628831868073474311732457023611118205047562399 86
2554340763841104243179081960393614316789486688479208550 20",
    "b":
"57449178641170524211735012111569259201994385842096936203494525793460895130970773585364715447870 23
6928935514185326246562922935412654108301295836689139071 40"
  },
  "encrypted_credential": {
    "a":
"55213734739350274658560961424964277188113441770640171504505958947739047817468824936099459131445 22
7013242700055299918863199419059232615752028134270098472 01",
    "b":
```

Figure 7.17: Before posting the ballot, the voter can take one final view of the overall ballot that will be cast to the bulletin board.

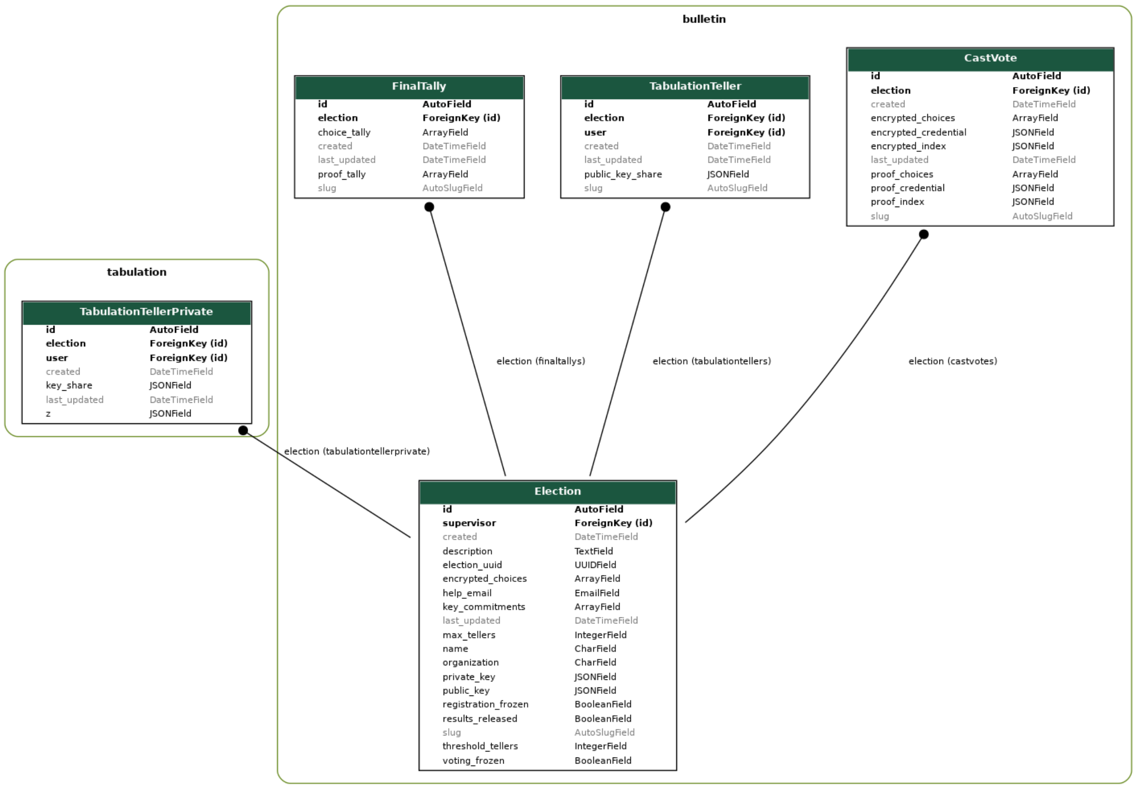


Figure 7.18: Each election has a single public key.

Chapter 8

Evaluation

8.1 By Number of Modular Exponentiations

In this section, we evaluate Resist's system by the number of modular exponentiations. It is compared against previously proposed coercion resistant protocols, namely JCJ/Civitas [12], AFT [18], and Selections [59].

In order to facilitate this comparison, we adopt the conventions that were used in a similar performance analysis by Clark et al. for Selection [59]. Assumptions include a single registration teller (registrar), T tabulation tellers (trustees), R registratered voters, and V_0 votes cast. Of the V_0 submitted votes, $V_1 \leq V_0$ votes have correct proofs, $V_2 \leq V_1$ are unique, and $V_3 \leq V_2$ have valid credentials.

Note here that, specifically for Selections [59], α represents the length of the cipertext list and β is the length of the voter's set of ciphertexts at the time of submitting a ballot. The performance measurements for Selections and AFT have been mentioned here directly as stated by the authors in their papers.

Selections and Civitas use traditional ElGamal decryption, while Resist and AFT use modified ElGamal. The modified version adds an additional modular exponentiation, $(g^m modp)$. However, in order to consider a uniform performance comparison, we use Clark et al.'s approach of using ElGamal accross the board. Given a traditional ElGamal cryptosystem, encryption requires 2 modular exponentiations, re-encryption adds 2 to that, and 1 exponentiation factors into decryption.

Registering for an election with a single registration teller requires a voter in Resist to perform a Needham Schroeder Lowe Protocol with the registration teller. The shares a similarity with Civitas in that Needham-Schroeder-Lowe is performed, credential shares are encrypted, re-encrypted, and both are sent for DVRP. The result is that the Registration Teller performs 7 modular exponentiations and the voter performs 4 additional exponentiations for verification.

AFT has a similar cost to both, Resist and Civitas. Selections depends on the acceptable length of ciphertext list, α (which usually around 5). Neither AFT nor Selections are orders of magnitude complex when compared to Resist or Civitas.

While casting a ballot, Resist requires to exponentiations each for encrypting a credential, a public index, and a choice. Generating a proof for their vote (VotePf), including their credentials and public index, requires 3 exponentiations. Proving that an encrypted choice is a 1-out-of-L re-encryption adds two modular exponentiations to it.

While JCJ, AFT, and Selections use a threshold cryptosystem, Civitas and Resist use a distributed cryptosystem. It is assumed here that a distributed system is a special case of a threshold system wherein T Tabulation Tellers are required in order to perform the tally.

In order to remove duplicates, all V_0 votes need to be raised to a shared random value. This operation is similar to reencrytion and adds 2 exponentiations for each voter. Thereafter, decrypting and adding all V_0 votes to a hash table will need 1 exponentiation for each T.

AFT's solution to removing duplicates is transferring this responsibility to external entities that can determine duplicates by observing the bulletin board. Selections, on the other hand, does not have a provision for casting duplicate votes.

The mix and verification of a mix used in Resist is the same as that used in Civitas. Each vote, V_2 passes 2 reencryptions as a result of two mixes by each tabulation teller.However, the list of accepted credentials need not be shuffled and reencrypted twice for 2 mixes.

Removing unregistered participants in Resist is different from Civitas in that it requires a single traversal through V_2 votes. Each Plaintext Equivalence Test requires $(8T + 1)$ exponentiations, given that Resist uses the Jakobsson and Juels algorithm.

During the Registration phase, Resist has the same number of exponentiations as Civitas. A similar pattern is observed in the Casting phase as well. These phases are not multiplied with the number of votes (say n) because they take place on individual systems, and the table shows the cost per vote.

Resist reduces the complexity of removing duplicate votes to linear time in the number of votes cast. As can be observed from the table, Civitas requires two V_1^2 traversals for this phase.

		JCJ/Civitas [12]	AFT [18]	Selections [59]	Resist
Registration	Registration Teller	7	9	2α	7
	Voter	11	10	4α-1	11
Casting	Voter	10	24	(2β+9)	11
Pre-Tally	Check Proofs	$4V_0$	$20V_0$	$(4\beta+6)V_0$	$5V_0$
	Remove Duplicates	$(1/2)(V_1^2 - V_1)(8T+1)$	—	—	$2V_0 + V_1T$
	Mix	$8V_2T + 4RT$	$20V_2T$	$12V_2T$	$8V_2T$
	Check Mix	$4V_2T + 2RT$	$10V_2T$	$6V_2T$	$4V_2T$
	Remove Unregistered	$(8T+1)V_2R$	$(16T+8)V_2$	$(8T+1)V_2$	$(8T+1)V_2$
	Check Removal	$(8T+1)V_2R$	$(16T+10)V_2$	$(8T+1)V_2$	$(8T+1)V_2$

Table 8.1: Comparison of Resist's Performance, measured by modular exponentiations

The mix used in Resist requires fewer exponentiations per mix by each tabulation teller in comparison to all three previous proposals.

When removing votes cast by unregistered voters, Civitas had a worst-case complexity of $\mathcal{O}(Nn)$ ($N = V_2$ and $n = R$). Resist brings this complexity down to $\mathcal{O}N$.

The comparisons in this section have primarily been made with JCJ/Civitas. While AFT and Selections are attemps at reducing the worst-case complexity and usability of JCJ/Civitas, there are known security vulnerabilities in their approaches. Resist, on the other hand, closely follows JCJ/Civitas, and its changes have been verified using ProVerif to not introduce new vulnerabilities in the system.

8.2 Timing Measurements

In order to capture performance measurements with respect to time taken, a Macintosh laptop with Firefox 67.0.4 was used. The laptop acted as a server and all client side interaction was experimented with on the localhost. The laptop is a 1.8GHz dual-core Intel Core i5 processor.

An election was simulated for a single question election with 2 choices. 500 votes were recorded in the overall process. A total of 3 tabulation tellers were responsible for mixes, 2 registration tellers generated credentials, while 10 voters were registered for the election.

Note that these values provide primitive idea of the time that the system can take. It is neither a best-case nor a worse-case scenario of the working of the system, since servers deployed for elections will vary widely.

Operation	**Unix Time**
Generating Distributed Election Key (Server), $\|p\| = 512$ bits	342 ms
Generating Credentials (Server)	413 ms
Encrypting a Ballot (Browser)	223 ms
Single Mix (Server)	668 ms
Mixnet (Server)	7 s
Decryption (Server)	83 s

Table 8.2: Timing Measurements

These values have not been evaluated against other systems owing to a disparity in the protocol, language, and system measurements. Similar time measurements can be found for Helios (which used a 2.2 GHz laptop with Firefox 2 in 2008) which did not perform mixes and for Civitas (which used a 3.0 GHz processor) which was evaluated on a 1 Gb LAN connection.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

Coercion resistance continues to be a major security caveat in both, physical and remote voting systems. Numerous protocols have been proposed to counter any influence on a voter while they participate in elections.

We have proposed a protocol for remote coercion resistant elections that can be tallied in linear time. This protocol has been verified to comply with a formal definition of coercion resistance. The verified protocol has been implemented as a web-based application. Both, the protocol and the system have been evaluated, and the protocol has been compared against previously proposed coercion resistant election protocols.

The protocol is primarily based on the JCJ Protocol because the security assumptions of the protocol have been verified satisfy the definition of coercion resistance. The literature covers suggestions to JCJ that attempt at either reducing its complexity or making its credentials readable. These suggestions, however, introduce vulnerabilities in the system. The thesis, therefore, builds on the oldest protocol and follows it closely. The changes made to handling of credentials and indices are verified to ensure that they satisfy the formal definition of coercion resistance.

The protocol addresses resistance to forced abstention attacks only after the registration phase. This forms the weakest link in the protocol. Once credentials have been obtained, however, the coercer is incapable of determining if a voter participated in the election.

Our work has been built upon suggested modifications to multiple existing protocols and systems. The implications of this modification have been validated and implemented. The system that has been developed can, as a result, be used in future attempts at countering coercion, and if possible, help voters be confident that their vote made a difference.

9.2 Limitations

Given the current status of the code powering Resist, an implementation of the system presents limitations, and in some cases, potential for extensions as described further.

Handling Credentials. Credentials in the implemented system are passed to the user as json files. This does not comply with recommended security practices. Since the credentials and credential shares in Resist are long integers, previous systems suggest delivering them as printed bar/QR codes. We leave this to future work.

Support for Write-ins. Tabulation tellers for Resist perform a homomorphic tally of all available votes on the system before decrypting the final sum. As a result of the nature of homomorphic encryption, the system cannot support additional choices that had not been defined at the start of the election. Therefore, the entry of answers or choices that do not already exist as a part of the system is not permissible in Resist. Additionally, the availability of such an option makes the system vulnerable to coercion, especially to a randomization attack, which has been described in Chapter 3.

Support for Rank-based Voting. Given the current model of its implementation, Resist only evaluates whether or not a choice has been made. It can be modified to include the total score that each candidate received on the basis of the ranks that they have been allotted. Given minimal inclusion, and an additional option at the time of creating an election, Resist will be able to support elections that demand rank-based voting.

9.3 Extensions

Support for Robustness Mixnets. As mentioned above, Resist has been designed such that tabulation tellers evaluate two re-encryption mixnets of the votes cast. If an implementation of the system has a an erraneous computation by any of the tellers or servers in the system, the mixnet needs to be restarted and the dishonest server needs to be eliminated. Robustness needs to be included in newer proposed protocols.

Benaloh Challenge. If voters doubt the integrity of their personal devices or browsers with regard to correct execution of encryption, they can verify this by auditing their ballots using a method suggested by Benaloh et al. [50]. The extension will include simple Javascript which can be made available along with other files that are fetched for casting a ballot. This, however, doesn't counter susceptible browsers on the user's end, in which case, Resist provides the option of posting ballots on a separate public repository of ballots for auditing.

Accessibility. Resist's front-end is eventually intended to replicate the suggestions from Civic Design's Anywhere Ballot [103, 104]. This includes providing options for colour contrasts, text size, and languages. Additionally, in order to work with independently developed screen readers, the html used should comply with recommendations in WAI ARIA [105].

Distribution. The Bulletin Board in Resist, like most election system implementations, is susceptible to DoS attacks [34]. Measures like distribution of source, or including multiple ballot boxes (as done in Civitas [14]), can mitigate these attacks but not counter them.

Bibliography

[1] M. Backes, C. Hritcu, and M. Maffei, "Automated verification of remote electronic voting protocols in the applied pi-calculus," in *2008 21st IEEE Computer Security Foundations Symposium*, June 2008, pp. 195–209.

[2] Ü. Madise and T. Martens, "E-voting in Estonia 2005. The first practice of country-wide binding Internet voting in the world," in *Electronic Voting 2006, 2nd International Workshop, Co-organized by Council of Europe, ESF TED, IFIP WG 8.5 and E-Voting.CC*, ser. Lecture Notes in Informatics, 2006, pp. 15–26.

[3] 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, no. 3, p. 497–505, 2009.

[4] M. Achieng and E. Ruhode, "The adoption and challenges of electronic voting technologies within the South African context," *CoRR*, vol. abs/1312.2406, 2013. [Online]. Available: http://arxiv.org/abs/1312.2406

[5] T. Fujiwara, "Voting technology, political responsiveness, and infant health: Evidence from Brazil," *Econometrica*, vol. 83, no. 2, pp. 423–464, 2015.

[6] S. Wolchok, E. Wustrow, J. A. Halderman, H. K. Prasad, A. Kankipati, S. K. Sakhamuri, V. Yagati, and R. Gonggrijp, "Security Analysis of India's Electronic Voting Machines," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866309

[7] Communications Security Establishment, "Cyber Threats to Canada's Democratic Process," Sep. 2018. [Online]. Available: https://cyber.gc.ca/en/

[8] E. National Academies of Sciences and Medicine, *Securing the Vote: Protecting American Democracy.* Washington, DC: The National Academies Press, 2018. [Online]. Available: https://www.nap.edu/catalog/25120/securing-the-vote-protecting-american-democracy

[9] J. M. Pleasants, *Hanging Chads: The Inside Story of the 2000 Presidential Recount in Florida.* New York: Palgrave Macmillan US, 2004. [Online]. Available: http://link.springer.com/10.1057/9781403973405

[10] D. Weigel, "Why are people giving Jill Stein millions of dollars for an election recount?" *The Washington Post*, Nov. 2016. [Online].

Available: https://www.washingtonpost.com/news/post-politics/wp/2016/11/24/why-are-people-giving-jill-stein-millions-of-dollars-for-an-election-recount/

[11] Press Trust of India, “JNUSU Poll: Protests, Violence Impedes Counting Process,” Sep. 2018.

[12] A. Juels, D. Catalano, and M. Jakobsson, “Coercion-resistant Electronic Elections,” in *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, ser. WPES ’05. New York, NY, USA: ACM, 2005, pp. 61–70. [Online]. Available: http://doi.acm.org/10.1145/1102199.1102213

[13] A. C. Myers, L. Zheng, S. Zdancewic, S. Chong, and N. Nystrom, “Jif: Java information flow,” *Software release. Located at http://www. cs. cornell.edu/jif*, 2005.

[14] M. R. Clarkson, S. Chong, and A. C. Myers, “Civitas: Toward a secure voting system,” in *Security and Privacy, 2008. SP 2008. IEEE Symposium on.* IEEE, 2008, pp. 354–368.

[15] D. Smith, “New cryptographic voting schemes with best-known theoretical properties,” in *FEE 2005 : Workshop on Frontiers in Electronic Elections*, 2005.

[16] S. G. Weber, R. Araujo, and J. Buchmann, “On coercion-resistant electronic elections with linear work,” in *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on.* IEEE, 2007, pp. 908–916.

[17] S. G. Weber, *Coercion-Resistant Cryptographic Voting: Implementing Free and Secret Electronic Elections.* VDM Publishing, 2008.

[18] R. S. dos Santos Araújo, “On remote and voter-verifiable voting,” Ph.D. dissertation, Technische Universität, 2008.

[19] R. Araujo, S. Foulle, and J. Traoré, “A practical and secure coercion-resistant scheme for remote elections,” in *Frontiers of Electronic Voting*, ser. Dagstuhl Seminar Proceedings, D. Chaum, M. Kutylowski, R. L. Rivest, and P. Y. A. Ryan, Eds., no. 07311. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2008. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2008/1295

[20] R. Araújo, N. B. Rajeb, R. Robbana, J. Traoré, and S. Youssfi, “Towards practical and secure coercion-resistant electronic elections,” in *International Conference on Cryptology and Network Security.* Springer, 2010, pp. 278–297.

[21] O. Spycher, R. Koenig, R. Haenni, and M. Schläpfer, “A new approach towards coercion-resistant remote e-voting in linear time,” in *International Conference on Financial Cryptography and Data Security.* Springer, 2011, pp. 182–189.

[22] M. H. Hansen, *The Athenian democracy in the age of Demosthenes: structure, principles, and ideology.* University of Oklahoma Press, 1999.

[23] I. McLean, H. Lorrey, and J. Colomer, "Voting in the medieval papacy and religious orders," in *International Conference on Modeling Decisions for Artificial Intelligence.* Springer, 2007, pp. 30–44.

[24] J. Guhin, "Democracy and the passions," *The Hedgehog Review*, vol. 19, no. 2, pp. 112–115, 2017.

[25] J. Sangster and L. Kealey, *Beyond the Vote: Canadian Women and Politics.* University of Toronto Press, 1989.

[26] Part 1 of the Constitution Act, "Canadian Charter of Rights and Freedoms," 1982.

[27] P. H. Russell, "The political purposes of the Canadian Charter of Rights and Freedoms," *Canadian Bar Review*, vol. 61, p. 30, 1983.

[28] R. Wright, *A People's Counsel: A History of the Parliament of Victoria 1856-1990.* Oxford University Press, 1992.

[29] S. Knack and M. Kropf, "Who uses inferior voting technology?" *PS: Political Science and Politics*, vol. 35, no. 3, p. 541–548, 2002.

[30] D. W. Jones, *The Evaluation of Voting Technology.* Boston, MA: Springer US, 2003, pp. 3–16. [Online]. Available: https://doi.org/10.1007/978-1-4615-0239-5_1

[31] J. P. Harris, "Data registering device," Mar. 15 1966, uS Patent 3,240,409.

[32] S. Popoveniuc and B. Hosp, "An Introduction to PunchScan." *Towards trustworthy elections*, vol. 6000, pp. 242–259, 2010.

[33] B. Adida, "Advances in cryptographic voting systems," Ph.D. dissertation, Massachusetts Institute of Technology, 2006.

[34] M. Bernhard, J. Benaloh, J. Alex Halderman, R. L. Rivest, P. Y. A. Ryan, P. B. Stark, V. Teague, P. L. Vora, and D. S. Wallach, "Public Evidence from Secret Ballots," in *Electronic Voting.* Cham: Springer International Publishing, 2017, pp. 84–109.

[35] P. H. O'Neill, "Elizabeth Warren Wants to Create a New Agency to Fix America's Laughable Election Security," *Gizmodo.* [Online]. Available: https://gizmodo.com/elizabeth-warren-wants-to-create-a-new-agency-to-fix-am-1835845432

[36] M. H. Weik, "A third survey of domestic electronic digital computing systems," Mar. 1961.

[37] D. W. Jones, *On Optical Mark-Sense Scanning.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 175–190. [Online]. Available: https://doi.org/10.1007/978-3-642-12980-3_10

[38] A. Kiayias, L. Michel, A. Russell, A. Shvartsman, U. V. Center, M. Korman, A. See, N. Shashidhar, and D. Walluck, "Security assessment of the diebold optical scan voting terminal," 2006. [Online]. Available: https://www.verifiedvoting.org/wp-content/uploads/2016/11/uconn_report-os.pdf

[39] B. Adida, "Convincing the Loser: Securing Elections against Modern Threats." Burlingame, CA: USENIX Association, Jan. 2019.

[40] R. Rahim, "Azmin: PKR party polls rife with problems, fair investigation needed - Nation | The Star Online." [Online]. Available: https://www.thestar.com.my/news/nation/2018/10/24/azmin-pkr-party-polls-rife-with-problems-fair-investigation-needed/

[41] N. Kersting and H. Baldersheim, *Electronic voting and democracy: A comparative analysis.* Palgrave Macmillan, London, 2004.

[42] B. Jacobs and W. Pieters, "Electronic voting in the netherlands: from early adoption to early abolishment," in *Foundations of security analysis and design V.* Springer, 2009, pp. 121–144.

[43] S. Kumar and E. Walia, "Analysis of electronic voting system in various countries," *International Journal on Computer Science and Engineering*, vol. 3, no. 5, pp. 1825–1830, 2011.

[44] T. S. James, "Fewer Costs, More Votes? United Kingdom Innovations in Election Administration 2000–2007 and the Effect on Voter Turnout," *Election Law Journal*, vol. 10, no. 1, pp. 37–52, 2011.

[45] B. B. Bederson, B. Lee, R. M. Sherman, P. S. Herrnson, and R. G. Niemi, "Electronic voting system usability issues," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '03. New York, NY, USA: ACM, 2003, pp. 145–152. [Online]. Available: http://doi.acm.org/10.1145/642611.642638

[46] D. L. Dill, B. Schneier, and B. Simons, "Voting and technology: Who gets to count your vote?" *Communications of the ACM*, vol. 46, no. 8, pp. 29–31, 2003.

[47] J. Franklin and J. Myers, "Interpreting babel: classifying electronic voting systems," in *5th International Conference on Electronic Voting 2012 (EVOTE2012)*, M. J. Kripp, M. Volkamer, and R. Grimm, Eds. Bonn: Gesellschaft für Informatik e.V., 2012, pp. 243–255.

[48] M. Germann and U. Serdült, "Internet voting and turnout: Evidence from Switzerland," *Electoral Studies*, vol. 47, pp. 1 – 12, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S026137941630453X

[49] P. Y. A. Ryan, P. B. Rønne, and V. Iovino, "Selene: Voting with Transparent Verifiability and Coercion-Mitigation," in *Financial Cryptography and Data Security*, J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 176–192.

[50] J. Benaloh, "Ballot casting assurance via voter-initiated poll station auditing." *EVT*, vol. 7, pp. 14–14, 2007.

[51] M. Hirt and K. Sako, "Efficient receipt-free voting based on homomorphic encryption," in *Advances in Cryptology — EUROCRYPT 2000*, B. Preneel, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 539–556.

[52] B. Adida, "Helios: Web-based Open-Audit Voting." in *USENIX security symposium*, vol. 17, 2008, pp. 335–348.

[53] 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

[54] P. Y. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia, "Prêt à voter: a voter-verifiable voting system," *IEEE transactions on information forensics and security*, vol. 4, no. 4, pp. 662–673, 2009.

[55] D. Chaum, "Secret-ballot receipts: True voter-verifiable elections," *IEEE security & privacy*, vol. 2, no. 1, pp. 38–47, 2004.

[56] R. Carback, D. Chaum, J. Clark, J. Conway, A. Essex, P. S. Herrnson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, A. T. Sherman, and P. L. Vora, "Scantegrity ii municipal election at takoma park: The first e2e binding governmental election with ballot privacy," in *Proceedings of the 19th USENIX Conference on Security*, ser. USENIX Security'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 19–19. [Online]. Available: http://dl.acm.org/citation.cfm?id=1929820.1929846

[57] B. Pfitzmann, "Breaking an efficient anonymous channel," in *Advances in Cryptology — EUROCRYPT'94*, A. De Santis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 332–340.

[58] J. Camenisch and A. Lysyanskaya, "Signature schemes and anonymous credentials from bilinear maps," in *Annual International Cryptology Conference.* Springer, 2004, pp. 56–72.

[59] J. Clark and U. Hengartner, "Selections: Internet voting with over-the-shoulder coercion-resistance," in *Financial Cryptography and Data Security*, G. Danezis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 47–61.

[60] ——, "Panic passwords: Authenticating under duress," in *Proceedings of the 3rd Conference on Hot Topics in Security*, ser. HOTSEC'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 8:1–8:6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496671.1496679

[61] C. J. F. Cremers, "The scyther tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification*, A. Gupta and S. Malik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 414–418.

[62] B. Blanchet, "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules," in *14th IEEE Computer Security Foundations Workshop (CSFW-14).* Cape Breton, Nova Scotia, Canada: IEEE Computer Society, Jun. 2001, pp. 82–96.

[63] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani *et al.*, "The AVISPA tool for the automated validation of internet security protocols and applications," in *International Conference on Computer Aided Verification.* Springer, 2005, pp. 281–285.

[64] K. Gjøsteen, "The Norwegian Internet Voting Protocol," in *E-Voting and Identity*, A. Kiayias and H. Lipmaa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–18.

[65] S. Delaune, S. Kremer, and M. Ryan, "Coercion-resistance and receipt-freeness in electronic voting," in *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, July 2006, pp. 12–42.

[66] ——, "Verifying Privacy-Type Properties of Electronic Voting Protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 435–487, 2009.

[67] ——, *Verifying Privacy-Type Properties of Electronic Voting Protocols: A Taster.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 289–309. [Online]. Available: https://doi.org/10.1007/978-3-642-12980-3_18

[68] S. Gerling, D. Jednoralski, and G. Xiaoyu, "Towards the verification of the civitas remote electronic voting protocol using proverif," in *Selected Topics of Information Security and Cryptography, Seminar WS*, Jul. 2008.

[69] A. Acquisti, "Receipt-free homomorphic elections and write-in ballots," *IACR Cryptology ePrint Archive*, vol. 2004, pp. 1–23, 2004.

[70] B. Meng, W. Huang, and D. Wang, "Automatic verification of remote internet voting protocol in symbolic model," *Journal of Networks*, vol. 6, no. 9, pp. 1262–1271, 2011.

[71] V. Cortier and C. Wiedling, "A formal analysis of the norwegian e-voting protocol," in *Principles of Security and Trust*, P. Degano and J. D. Guttman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 109–128.

[72] M. Backes, M. Maffei, and D. Unruh, "Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol," in *IEEE Symposium on Security and Privacy, 2008. SP 2008.* IEEE, 2008, pp. 202–215.

[73] M. Abadi, B. Blanchet, and C. Fournet, "Just fast keying in the pi calculus," *ACM Transactions on Information and System Security*, vol. 10, no. 3, Jul. 2007. [Online]. Available: http://doi.acm.org/10.1145/1266977.1266978

[74] T. Y. Woo and S. S. Lam, "A semantic model for authentication protocols," in *Research in Security and Privacy, 1993. Proceedings., 1993 IEEE Computer Society Symposium on.* IEEE, 1993, pp. 178–194.

[75] G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using FDR," in *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 1996, pp. 147–166.

[76] B. Warinschi, "A computational analysis of the Needham-Schroeder-(Lowe) protocol," in *16th IEEE Computer Security Foundations Workshop, 2003. Proceedings.* IEEE, 2003, pp. 248–262.

[77] J. Benaloh and D. Tuinstra, "Receipt-free secret-ballot elections," in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing.* ACM, 1994, pp. 544–553.

[78] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[79] 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, Feb. 1978. [Online]. Available: http://doi.acm.org/10.1145/359340.359342

[80] K. S. McCurley, "The discrete logarithm problem," in *Proceedings of the Symposium in Applied Mathematics*, vol. 42. USA, 1990, pp. 49–74.

[81] E. B. Barker, "Digital Signature Standard (DSS)," Tech. Rep., 2013.

[82] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, July 1985.

[83] R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack," in *Advances in Cryptology — CRYPTO '98*, H. Krawczyk, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 13–25.

[84] J. Herzog, M. Liskov, and S. Micali, "Plaintext awareness via key registration," in *Annual International Cryptology Conference.* Springer, 2003, pp. 548–564.

[85] P. Ribenboim, *The Little Book of Bigger Primes.* Springer Science & Business Media, 1991.

[86] D. Shanks, "Class number, a theory of factorization, and genera," in *Proceedings of the Symposium on Pure Mathematics*, vol. 20, 1971, pp. 41–440.

[87] L. M. Adleman, "The function field sieve," in *International Algorithmic Number Theory Symposium.* Springer, 1994, pp. 108–121.

[88] L. Adleman, "A subexponential algorithm for the discrete logarithm problem with applications to cryptography," in *20th Annual Symposium on Foundations of Computer Science (SFCS 1979).* IEEE, 1979, pp. 55–60.

[89] S. Halevi and H. Krawczyk, "Strengthening digital signatures via randomized hashing," in *Annual International Cryptology Conference.* Springer, 2006, pp. 41–59.

[90] C. P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, Jan 1991. [Online]. Available: https://doi.org/10.1007/BF00196725

[91] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Advances in Cryptology — CRYPTO' 92*, E. F. Brickell, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 89–105.

[92] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand, "A guide to fully homomorphic encryption." *IACR Cryptology ePrint Archive*, vol. 2015:1192, pp. 1–35, 2015.

[93] M. Jakobsson, A. Juels, and R. L. Rivest, "Making mix nets robust for electronic voting by randomized partial checking," in *Proceedings of the 11th USENIX Security Symposium.* Berkeley, CA, USA: USENIX Association, 2002, pp. 339–353. [Online]. Available: http://dl.acm.org/citation.cfm?id=647253.720294

[94] 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. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4460080506

[95] M. Jakobsson and A. Juels, “Mix and match: Secure function evaluation via ciphertexts,” in *Advances in Cryptology — ASIACRYPT 2000*, T. Okamoto, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 162–177.

[96] A. Holovaty and J. Kaplan-Moss, *The definitive guide to Django: Web development done right*. Apress, 2009.

[97] B. Momjian, *PostgreSQL: introduction and concepts*. Addison-Wesley New York, 2001, vol. 192.

[98] M. Sanner, “Python: a programming language for software integration and development,” *Journal of molecular graphics and modelling*, vol. 17, no. 1, p. 57—61, February 1999. [Online]. Available: http://europepmc.org/abstract/MED/10660911

[99] S. O’Grady, “The RedMonk Programming Language Rankings: January 2019,” Mar. 2019. [Online]. Available: https://redmonk.com/sogrady/2019/03/20/language-rankings-1-19/

[100] D. Flanagan, *JavaScript: the definitive guide*. O’Reilly Media, Inc., 2006.

[101] D. Mitropoulos, P. Louridas, V. Salis, and D. Spinellis, “Time present and time past: Analyzing the evolution of javascript code in the wild,” in *Proceedings of the 16th International Conference on Mining Software Repositories*, ser. MSR ’19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 126–137. [Online]. Available: https://doi.org/10.1109/MSR.2019.00029

[102] M. Zadka and G. v. Rossum.

[103] D. Chisnell, D. Davies, and K. Summers, “Any device, anywhere, any time: A responsive, accessible ballot design. the information technology and innovation foundation,” The Information Technology and Innovation Foundation, Tech. Rep., Jul. 2013. [Online]. Available: http://elections.itif.org/reports/AVTI-007-Chisnell-Davies-Summers-2013.pdf

[104] K. Summers, D. Chisnell, D. Davies, N. Alton, and M. Mckeever, “Making voting accessible: Designing digital ballot marking for people with low literacy and mild cognitive disabilities,” in *2014 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 14)*. San Diego, CA: USENIX Association, 2014. [Online]. Available: https://www.usenix.org/conference/evtwote14/workshop-program/presentation/summers

[105] W. W. W. Consortium, “Accessible rich internet applications (WAI-ARIA) 1.0,” 2014.