

2-17-2026 1:45 PM

Autonomous Election Verification with Zeeperio

Aikamdeep Malhotra

Supervisor: Essex, Aleks, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Engineering
Science degree in Electrical and Computer Engineering

© Aikamdeep Malhotra 2026

Abstract

This thesis presents Zeeperio, a verifiable voting protocol that makes end-to-end election verification routine and automatic. Zeeperio uses Eperio's election table structure and replaces the cut-and-choose auditing with a custom succinct zk-SNARK proof. This proof shows that all ballots are properly accounted for and that the published outcome is correct. The proof is submitted to an Ethereum smart contract for public, automatic verification, so an incorrect tally is rejected without relying on voluntary auditors. We implemented the protocol, and evaluated a 100,000-ballot, 5-candidate election. The proofs resulted in an on-chain verification cost of ~4.2 million gas, which translates to ~\$30 USD in fees on Ethereum at the measured prices. These results show that large scale public verification can be inexpensive and compatible with existing paper-election workflows.

Keywords E2E verifiable voting, zk-SNARKs, election verification, Ethereum, smart contracts, cryptographic proofs, Polynomial-IOPs, zero-knowledge proofs

Summary for Lay Audience

This thesis is about making election result checking routine, not optional. In many elections, the public must largely trust that the reported totals are correct. Cryptographic end-to-end verifiable voting systems allow for voters and observers to check an election, but the checking process is often dependent on volunteers and specialized tools, so it may not happen consistently.

We present *Zeeperio*, a protocol that produces a short cryptographic proof showing that the published election outcome matches the recorded ballots. Instead of asking people to manually audit large datasets, *Zeeperio* generates a compact proof that can be verified automatically. The key idea is to have a public smart contract verify the proof on the Ethereum blockchain. If the election data is inconsistent or the tally is wrong, the proof fails and the contract rejects it. This makes verification automatic and publicly visible, while still allowing independent auditors to check the same proof themselves.

We implemented *Zeeperio* and evaluated it on an election with 100,000 ballots and 5 candidates. The proofs are very small and verifying it on Ethereum's test network cost under \$30 USD (at the time). Compared to prior approaches that require very large proofs and heavy computation, these results show that large-scale public verification can be practical and inexpensive.

Table of Contents

Abstract	ii
Summary for Lay Audience	iii
Table of Contents	iv
List of Tables	x
List of Figures	xi
1 Introduction	1
Need for Automated Election Verification	1
Limitations of current E2E-V systems	2
1.1 Overview of <i>Zeeperio</i> and Contributions	3
1.2 Thesis Outline	5
2 Literature Review, Gap, and Problem Statement	7
2.1 Foundations of Secure E2E-V Voting Protocols	7
Cast-as-Intended	7
Recorded-as-Cast	8
Counted-as-Collected	9
2.2 Privacy and Security	9
Ballot Secrecy	9
Receipt Freeness	10
Coercion Resistance	10
2.3 A Review of Existing E2E-V Voting Protocols	11
2.3.1 Eperio	11
Verifiability	12

	Privacy and Security	13
	Usability	13
2.3.2	Scantegrity II	13
	Cryptography and Verifiability	14
	Privacy and Coercion Resistance	15
	Trust Model	15
	Usability	16
2.3.3	Civitas	17
	Verifiability	17
	Coercion Resistance	18
	Privacy	18
	Trust Model	19
2.3.4	Helios	19
	Tally Verification	20
	Privacy and Limitations	21
	Usability	21
2.3.5	STAR-Vote	23
	Cryptography and Verifiability	23
	Security	24
	Privacy	25
	Trust Model	26
	Usability	26
2.3.6	ElectionGuard	27
	Verifiability	27
	Privacy	28
	Trust Model	29
	Usability	30

2.3.7	Voatz	31
	Verifiability	31
	Security Vulnerabilities	31
	Privacy	32
	Trust Model	33
	Deployment	33
2.3.8	Comparative Analysis	34
2.4	Gap and Problem Statement	34
2.4.1	Realpolitik vs. Realcryptik: A Framework for Protocol Design . . .	37
	Realpolitik	37
	Realcryptik	37
2.4.2	Zeeperio: A New Compromise for Automated Verifiability.	38
3	Voting in Canada	40
3.1	Legal and Constitutional Framework	40
3.1.1	Supreme Court - Public Confidence	40
3.1.2	Opitz v. Wrzesnewskyj	41
3.1.3	The Municipal Elections Act of Ontario (MEA)	41
3.1.4	Principles vs Reality	42
3.2	From Observation to Blind Trust	43
3.2.1	Manually Counted Paper Ballots	43
3.2.2	Optical Scan Tabulators	44
3.2.3	Municipal Online Voting Black Box	44
3.3	Black Box Security Analysis	45
3.3.1	2018 Bandwidth Crisis	45
3.3.2	Side-Channel Attacks	46
3.3.3	Lack of Auditability	46
3.3.4	Authentication	47

3.4	A Review of Compliance	47
3.5	Cryptographic Solution	48
3.5.1	Bridging the Gap	48
3.5.2	Zeeperio	49
3.6	The Faith in Technology Bias	50
3.7	Comparative Analysis of Canadian Voting Methods	50
4	Mathematical Foundations	52
4.1	Finite Fields and Polynomials	52
4.1.1	Finite Fields	52
4.1.2	Polynomials	53
	Vanishing Polynomials	53
	Lagrange Basis	53
	Schwartz-Zippel Lemma	54
4.2	Groups, Elliptic Curves, and Pairings	55
4.2.1	Hardness assumptions	56
4.2.2	Elliptic curve groups	57
4.2.3	Bilinear Pairings	58
4.2.4	Algebraic Group Model (AGM)	59
4.3	Hash Functions and the Fiat-Shamir Transformation	60
4.3.1	Hash Functions	60
4.3.2	Fiat-Shamir Heuristic	60
4.4	Commitment Schemes	61
4.4.1	Pedersen Commitment Scheme	63
4.4.2	KZG Commitment Scheme	64
4.5	SNARKs	67
4.5.1	Interactive proofs and arguments	67
4.5.2	Polynomial Interactive Oracle Proofs (Poly-IOPs)	69

4.5.3	SNARKs: Succinct Non-Interactive Arguments of Knowledge . . .	70
	SNARK families	71
5	Zeeperio: Verifying Elections using Poly-IOP Gadgets and On-Chain Veri-	
	fication	72
5.1	Notation and Preliminaries	72
5.1.1	Polynomial IOPs and KZG Commitments	73
5.1.2	Constraint Enforcement using Vanishing Polynomials	76
5.1.3	On-Chain Verification via Ethereum	76
5.1.4	Finite Field and Evaluation Domain	78
5.1.5	Zero-Knowledge	79
5.2	Setup Phase	80
5.3	Audit Column Constraints	81
5.3.1	Padding	82
5.3.2	Values	82
5.3.3	Blocks	83
5.3.4	Count	85
5.4	Constraint: Mark Column	86
5.5	Constraint: Audit and Mark Column Exclusions	89
5.6	Constraint: Tally Check	90
5.7	Election Verification	92
5.8	Constraint: Voter Verification	94
5.8.1	Print Audit	94
5.8.2	Voter Checks	94
	Ballot Inclusion	95
5.8.3	Receipt Correctness	101
6	Zeeperio Proofs of Security, Implementation, Evaluation, and Future Work	103

6.1	Proofs of Security	103
6.1.1	Security Assumptions	105
6.1.2	Soundness: Zeeperio Constraints \rightarrow Eperio Conditions	106
6.1.3	Completeness: Eperio Conditions \rightarrow Existence of Proof	112
6.2	Implementation	117
6.2.1	Prover Overview	118
	CLI and Proof Generation	118
	Calldata Optimization	121
6.2.2	Verifier Overview	121
	Verifier Smart Contract Design	121
	Inputs and Parsing	122
	Recomputing Challenges	122
	Pairing Checks via Precompiles	122
	Foundry	123
	Sepolia Testnet	124
6.3	Results and Discussion	124
	Performance	125
	Mainnet Feasibility	126
	Comparison with other SNARKs and STARKs	126
6.4	Future Work	132
	References	135

Curriculum Vitae **152**

List of Tables

2.1	Comparison of various E2E-V voting protocols across key properties. . . .	35
3.1	Security and Legal Analysis of Canadian Voting Methods	51
6.1	Fixed sample election instance used for benchmarking.	125
6.2	Zeeperio benchmark with sample election. All measurements are from a single-core execution on standard hardware (MacBook M3 Pro).	125
6.3	Zeeperio compared against per-ballot GP-SNARK and ZK-STARK ap- proaches.	128

List of Figures

2.1	Each ballot contains a unique serial number, a randomized candidate list, and a perforation between the optical scan ovals and the candidate list. Once the voter marks their choice, they tear off and keep only the receipt portion showing the position of their mark. Each optical scan oval, its mark-state, and the corresponding candidate name are recorded in a randomly assigned row [1].	12
2.2	A Scantegrity II style ballot is shown with invisible regions indicated via a false-color overlay.	14
4.1	Multi-party trusted setup (MPC) for KZG.	65
5.1	Example election data representation with three candidates	73
5.2	Three layers of abstraction for constructing succinct proofs. From left to right, the prover’s statement is expressed at (i) the VM level (zkVM), (ii) the circuit level, or (iii) directly as Poly-IOP gadgets. Shifting right is “closer to the metal” as it exposes more algebraic structure to the proof system, reducing compilation overhead and constraint inflation, and typically yielding more efficient proving and cheaper verification.	75
5.3	Poly-IOP structure	77

Chapter 1

1 Introduction

Elections are the foundation of democracy, and their integrity is essential to maintaining public trust. End-to-end verifiable (E2E-V) voting protocols aim to protect that integrity by allowing voters and observers to independently verify that ballots are cast, recorded, and tallied correctly, all without having to rely on the election authority's honesty [2]. Over decades of research, cryptographers have developed many voting schemes that promise both strong integrity and ballot secrecy. Pilot deployments have shown that E2E-V systems can work in practice, at least on a small scale [3]. Yet the full ideal of verifiability, where anyone can mathematically confirm the correctness of an election result, remains far from common in real-world elections. In most cases, public verification is minimal or nonexistent, and any cryptographic proofs that are produced often go unchecked. When verification does occur, it is usually an optional, ad hoc effort performed by a handful of interested experts rather than a standard step in the election process. This persistent gap between cryptographic theory and real-world practice motivates the research in this thesis.

Need for Automated Election Verification

One major problem facing verifiable elections lies in its usability and automation. Existing E2E-V protocols, such as Scantegrity II and Helios, offer mechanisms for public verification, but they often depend on voters manually checking receipts or on auditors running specialized verification tools [4]. The Eperio protocol simplified Scantegrity's approach by allowing verification through software like Microsoft Excel [1]. Even so, it still relied on election authorities to coordinate a verification effort and hope that *someone* would actually perform it. In real-world scenarios, election officials have neither the incentive nor the means to ensure that verifiers participate, leaving many valid proofs unexamined and undermining the goal of universal verifiability.

Automated verification can solve this problem. If the verification process is mandatory and machine-checked, every proof of correctness would be validated automatically. Recent advances in succinct zero-knowledge proofs (zk-SNARKs) make this vision feasible. zk-SNARKs can attest to complex computations (like tallying encrypted votes) using a tiny proof that can be verified quickly and cheaply. With this technology, a smart contract can verify an election’s correctness on behalf of the public, which allows the system itself to become the verifier [5]. Once the election data is put into a succinct proof, anyone can check its integrity instantly. This thesis explores how zk-SNARKs can make E2E verification not only possible, but routine, automated, and transparent.

Limitations of current E2E-V systems

Despite their promise, current verifiable voting systems face practical challenges that limit adoption. Many rely on complex cryptographic components that are computationally heavy and difficult to scale. Approaches like Eperio reduced complexity using aggregated commitments and simple verification tools, but still required effort proportional to the number of ballots, which becomes infeasible for large elections.

Recent research integrating SNARKs or STARKs into voting systems has achieved stronger security guarantees, but often at a high cost. Proofs can be several megabytes in size, and generating them may take hours for even mid-sized elections. These performance constraints make such systems impractical for real-time or on-chain verification [6]. Additionally, some E2E-V schemes trade away privacy or security for convenience, relying on assumptions about honest election authorities rather than providing cryptographic guarantees. These limitations show the need for a new approach, one that preserves strong verifiability and voter privacy while achieving efficiency and seamless integration into real election workflows.

1.1 Overview of Zeeperio and Contributions

In this thesis, we present Zeeperio, a new verifiable voting protocol that addresses the above challenges by combining E2E-V voting with modern zk-SNARK technology. Zeeperio can be seen as a modern update to the Eperio protocol, designed to make election verification automatic. The core idea is to replace Eperio’s *cut-and-choose* proof mechanism with a custom succinct argument that proves the correctness of the election tally. Instead of producing numerous open commitments that must be manually audited, the election authority produces a single zk-SNARK proof showing that all ballots are properly accounted for. This proof is then published to a blockchain smart contract, which automatically verifies it, thereby publicly confirming the election outcome. By using a smart contract, Zeeperio turns the verification step into a *mandatory* part of the process. An incorrect result would cause the proof verification to fail and be caught immediately by the system, all without relying on human auditors.

Zeeperio introduces several novel contributions to the state of E2E-V voting:

- **Automated Verifiability:**

Zeeperio enables *hands-off* verification of election results. A deployed smart contract can accept a proof of the election outcome and check if it is valid on-chain, giving immediate feedback on whether the tally is correct. This guarantees that verification is performed **every time**, unlike prior systems where it was optional. Election authorities and the public gain automatic assurance that the outcome has been independently validated, moving verifiability from an ad hoc task to a standard procedure.

- **Succinct Proofs:**

The protocol uses a custom zk-SNARK built on a *Polynomial Interactive Oracle Proof*, tailored specifically to the Eperio election setting. As a result, Zeeperio produces constant-size proofs that remain small even as the number of ballots grows. Verifying

the proof requires only a few milliseconds and a few cryptographic operations on-chain, making it cheap enough for blockchain execution. We implemented Zeeperio and demonstrated that it can handle a large election (100,000 ballots with 5 candidates) with proof sizes of only a few kilobytes, and on-chain verification costing under \$30 USD. These are orders-of-magnitude improvements over previous approaches. Thus, Zeeperio’s efficiency directly translates to lower costs and better scalability for real elections.

- **Verifiability Guarantees:**

Zeeperio maintains the verifiability guarantees of E2E-V systems. Individual verifiability is preserved (each voter can still check that their vote is included on the bulletin board), and universal verifiability is achieved through the publicly checkable SNARK proof. The protocol upholds counted-as-cast integrity by proving that the published tally matches the collected ballots. At the same time, Zeeperio addresses specific limitations of the original Eperio protocol. It includes constraints to ensure no ballot is counted twice or overvoted (something Eperio did not explicitly guarantee) and can support ballot styles with confirmation codes (like Scantegrity II). Thus, Zeeperio improves both the verifiability and practicality of the voting process.

- **Integration with Existing Voting Paradigms:**

Rather than reinventing the voting process, Zeeperio is designed to be a drop-in enhancement to the Eperio/Scantegrity-style workflows. It works with ballots that are collected in an open commitment form (where encrypted votes are posted to a public bulletin board) much like existing optical-scan or E2E-V systems, relying on legal/procedural safeguards to protect ballot secrecy in practice. This conscious design choice (in line with the *Realcryptik* philosophy) means that election officials can adopt Zeeperio’s verification mechanism without needing to overhaul how votes are cast or stored. The protocol simply adds an automated verification layer on top of

the traditional process.

Zeeperio contributes a new design for verifiable voting where cryptographic proofs and smart contracts replace manual auditing. It demonstrates that with careful protocol design, one can achieve practical, scalable election verification that is both universally verifiable and autonomously enforced. The implementation and evaluation of Zeeperio, as detailed later in this thesis, validate that these theoretical benefits hold up in practice.

1.2 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 provides a literature review of secure voting protocols and identifies the gap between existing E2E-V solutions and real-world election practices. We discuss foundational concepts (cast-as-intended, recorded-as-cast, counted-as-collected) and examine why current systems have not fully achieved transparent, automated verification. The chapter ends with the problem statement, motivating an automated verifiability approach.

Chapter 3 examines the current state of elections in Canada, and more specifically in Ontario. We review the legal and operational landscape of voting, and highlight how *Realpolitik* considerations have led to the adoption of systems with known verification weaknesses. This analysis sets the stage for our proposed solution by illustrating the need for a more *Realcryptik* approach that does not compromise on cryptographic integrity.

Chapter 4 covers the necessary mathematical foundations and cryptographic preliminaries for the Zeeperio protocol. We introduce finite fields, elliptic curves, and polynomial commitment schemes (with a focus on KZG commitments) as well as zero-knowledge proof basics relevant to Zeeperio. This background equips the reader to understand the technical design of Zeeperio in subsequent chapters.

Chapter 5 presents the design of Zeeperio in detail. We describe the election model and

notation, then outline how the *Polynomial Interactive Oracle Proof* framework is used to encode the election as polynomial identity constraints. The chapter explains Zeeperio’s constraint system, and how these are compiled into a zk-SNARK. We also discuss the on-chain verification contract logic.

Chapter 6 provides the security analysis, implementation details, evaluation, and future work of Zeeperio. We formally argue the soundness and completeness of Zeeperio’s proofs (showing that a cheating authority cannot forge a valid proof for an incorrect outcome, under standard cryptographic assumptions). The implementation section describes our prototype built using Ethereum’s BN254 pairing precompiles for on-chain proof verification. We then report experimental results, showing Zeeperio’s performance on a large election dataset and comparing it with other approaches. Key results include the proof size and generation time for 100,000 ballots and the cost of on-chain verification, showing the practicality of our approach. Finally, we discuss future work. We reflect on the implications of making election verification autonomous and outline potential improvements that could be pursued to further bridge the gap between cryptographic election guarantees and real-world deployment.

By following this structure, the thesis progresses from foundational concepts and real-world context to Zeeperio’s novel contributions. It then concludes by showing the work’s impact and outlining directions for future extensions.

Chapter 2

2 Literature Review, Gap, and Problem Statement

This chapter reviews the literature on secure voting protocols, identifies the gaps in existing approaches, and addresses the problem statement of the thesis. We begin by reviewing the foundations of end-to-end verifiable (E2E-V) voting protocols and explain the verifiability properties and privacy requirements these systems must satisfy.

2.1 Foundations of Secure E2E-V Voting Protocols

The goal of end-to-end verifiable (E2E-V) voting protocols is to guarantee the integrity of an election that can be *verified* by voters and observers alike, all without relying on a central authority [7–9]. E2E verifiability requires two levels of verification:

- **Individual Verifiability:** Each voter can verify their own vote was correctly included in the election.
- **Universal Verifiability:** Any observer can verify the overall election outcome is correct based on the recorded votes [10].

To achieve E2E verifiability, we require three core properties: *cast-as-intended*, *recorded-as-cast*, and *counted-as-collected* [11]. Together, these properties give confidence to voters that their vote was captured and counted correctly in the election. If any error or tampering occurs at any stage of the election, it will be detected by the voter or auditors.

Cast-as-Intended

A voting protocol satisfies this property if every voter can independently confirm whoever they *intended* to vote for is accurately reflected on their ballot [12]. For electronic voting protocols, this translates to their ballot being correctly constructed by the software or voting

machine. This is especially important in scenarios where the voting machine may be compromised or malicious. In such cases, the machine might display *Candidate A* on the screen, but actually encrypt a vote for *Candidate B*. We describe this property formally as follows:

Definition 1. Let \mathcal{V} be the voter's intended vote and \mathcal{B} be the voter's ballot. A voting protocol provides *cast-as-intended verifiability* iff there exists a verification process Verify_{CAI} such that:

$$\Pr[\text{Verify}_{CAI}(\mathcal{B}, \mathcal{V}) = 1 \mid \mathcal{B} \text{ encodes } \mathcal{V}] \geq 1 - \text{negl}(\lambda)$$

$$\Pr[\text{Verify}_{CAI}(\mathcal{B}, \mathcal{V}) = 1 \mid \mathcal{B} \text{ does not encode } \mathcal{V}] \leq \text{negl}(\lambda)$$

where λ is the security parameter and $\text{negl}(\lambda)$ denotes a negligible function.

Recorded-as-Cast

A voting protocol satisfies this property if every voter can independently confirm that their cast ballot has been correctly *recorded* in the public election record. In typical E2E-V voting protocols, this is achieved by providing the voter with a receipt that contains a *tracking code* for their ballot that can be used to check a public list of ballot records known as the *public bulletin board* [13]. The bulletin board must be *publicly verifiable*. This ensures once a ballot is recorded, it cannot be deleted or modified without detection [14]. If the code appears in the bulletin board, the voter is assured their vote is included in the election. We describe this property formally as follows:

Definition 2. Let \mathcal{BB} denote the public bulletin board and r be the voter's receipt. A voting protocol provides *recorded-as-cast verifiability* iff:

$$r \in \mathcal{BB} \rightarrow \exists \mathcal{B} \in \mathcal{BB} \text{ s.t. } H(\mathcal{B}) = r$$

where H is a cryptographic hash function.

Counted-as-Collected

A voting protocol satisfies this property if any observer can verify that all collected ballots are *correctly included* in the final tally. In typical E2E-V protocols, this is achieved by using cryptographic tools such as homomorphic encryption (where encrypted values can be summed without decryption), or mix-nets (where ballots are shuffled and re-encrypted). The protocol then provides some proofs to show the correctness of the tally, all without revealing individual votes [15]. We describe this property formally as follows:

Definition 3. Let \mathcal{T} denote the published tally, $\mathcal{BB} = \{\mathcal{B}_1, \dots, \mathcal{B}_n\}$ be the set of ballots on the bulletin board, and π be a cryptographic proof of correct tallying. A voting protocol provides counted-as-collected verifiability iff there exists a publicly computable function Verify_{CAC} such that:

$$\text{Verify}_{CAC}(\mathcal{T}, \mathcal{BB}, \pi) = \text{true} \text{ s.t. } \mathcal{T} = \text{Tally}(\mathcal{BB})$$

2.2 Privacy and Security

Along with verifiability, E2E-V voting protocols must also satisfy strong privacy and security properties to protect voters against vote-selling and coercion. These core properties are *ballot secrecy*, *receipt freeness*, and *coercion resistance*. Each of these notions is important for free and fair elections.

Ballot Secrecy

Ballot secrecy (also known as voter anonymity or privacy) requires that no party learns how an individual voter voted, beyond what can be inferred from the overall outcome. In this notion, the adversary is assumed to be *passive*. It should not be able to link a cast ballot back to its voter from the observable. Josh Benaloh, a senior cryptographer at

Microsoft Research emphasized "A secret ballot requires that . . . no voter's choices can be surreptitiously released or inferred. The goal of a secret ballot is to provide an election free from undue influence" [16]. Ballot secrecy is implemented by encrypting or anonymizing ballots, and ensuring any published results do not reveal individual vote contents. However, ballot secrecy alone does not prevent coercion or vote-selling, as a voter might still be able to prove their vote to a third party.

Receipt Freeness

Receipt freeness is a stronger notion that requires a voter to not be able to obtain any proof or *receipt* that would credibly show how they voted to a third party. Here, the assumption is that the adversary is post-factum, where it can *check in* with the voter after the election and ask for evidence of how they voted. This property was first introduced by Benaloh and Tuinstra to prevent vote-selling and coercion [17]. Ron Rivest, a pioneer in cryptographic voting, emphasized "the voter must not be given a 'receipt' that would allow them to prove how they voted to someone else-otherwise the voter could be coerced or bribed" [18]. Receipt freeness is implemented by ensuring that any data a voter can retain does not allow them to prove how they voted. Additionally, it ensures no voter has access to the secret *randomness* used to encrypt their vote. However, receipt freeness does not protect against more powerful coercion attacks where the adversary can pressure the voter during the voting process itself.

Coercion Resistance

Coercion resistance is the strongest of the three properties, requiring even when a voter is coerced or bribed during the voting process, they can still vote freely without the coercer being able to detect it. In this notion, the adversary is *interactive* and *omnipresent*, where it can stand over the voter's shoulder, provide the encryption keys itself, or force the voter to share their screen. This property was introduced by Juels, Catalano, and Jakobsson to model

strong coercion threats. They stated "We define a scheme to be coercion-resistant if it is infeasible for the adversary to determine whether a coerced voter complies with the demands [even when the adversary provides the keys or watches the process]" [19]. Coercion-resistant protocols are implemented by giving voters *deniable* ways to interact with the system. This can include issuing fake, but indistinguishable credentials or alternative transcripts so that any behaviour observed by the coercer cannot be linked to the voter's true intent.

Most E2E-V protocols cannot achieve full coercion resistance in the strongest sense, especially internet voting schemes. In practice, many protocols aim for weaker threat assumptions or combine E2E verifiability with procedural safeguards such as supervised polling places to reduce real-world risk [20]. Even though full coercion resistance is impractical, it remains as a valuable *design benchmark* that guides how protocols handle voter influence, how much information they expose to observers, and what assumptions they make about the voting environment.

2.3 A Review of Existing E2E-V Voting Protocols

We now review several E2E-V voting protocols, describing their design, cryptographic foundations, trust assumptions, usability, and deployment, with focus on how they achieve the properties discussed above. As each protocol balances these notions differently, it is important to understand their mechanisms and trade-offs, which is key to the literature review. We analyze each protocol in detail below.

2.3.1 Eperio

Eperio is an E2E-V *paper-based* voting protocol that is designed for simplicity and auditability, introduced by Essex et al. [1]. As shown in Figure 2.1, the candidate list is randomized. After the voter marks their ballot, they *shred* the candidate list and keep only the receipt portion showing the position of their mark.

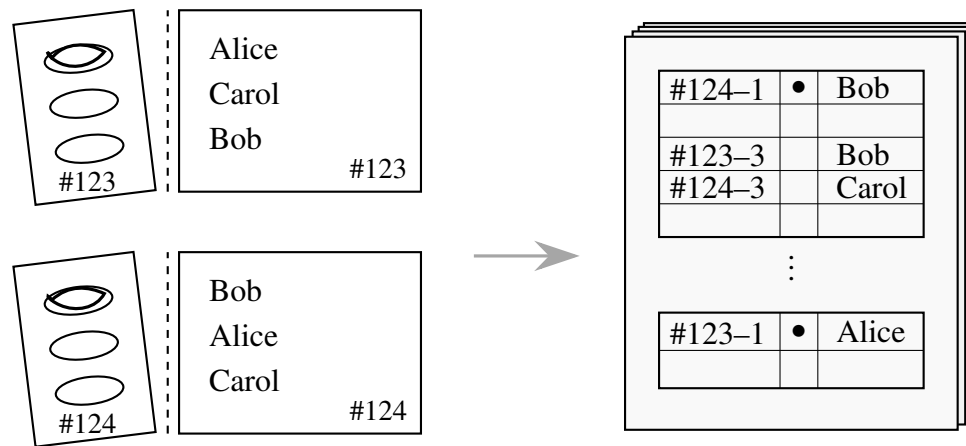


Figure 2.1: Each ballot contains a unique serial number, a randomized candidate list, and a perforation between the optical scan ovals and the candidate list. Once the voter marks their choice, they tear off and keep only the receipt portion showing the position of their mark. Each optical scan oval, its mark-state, and the corresponding candidate name are recorded in a randomly assigned row [1].

Behind the scenes, Eperio arranges all ballots into a cryptographic table (similar to the concept of a shuffle ballot box). This table contains columns for the commitments of the ballots' serial numbers, their mark positions, and the commitments for all the candidates listed. This table is shuffled randomly so one cannot link a ballot to its chosen vote. After the election, the election authorities (EA) partially reveal columns on a public bulletin board as a proof of consistency and correctness [1].

Verifiability

Each voter can verify *cast-as-intended* and *recorded-as-cast* by checking if their receipt's serial number and marked position appear in the posted data on the bulletin board. *Counted-as-collected* is also demonstrated by verifying the table with a spreadsheet for tally correctness. One of Eperio's main objectives was to make the verification process very simple for lay auditors. Eperio uses spreadsheets and standard file encryption to verify the outcome of the election. This avoids the need for specialized software or expertise [1].

Privacy and Security

Since the candidate list is randomized for each ballot, and the voter destroys their ballot's candidate list, there is no information that is revealed about the chosen candidate. The commitments hide the votes' values, so ballot secrecy is preserved. Vote privacy holds under the assumption that at least one election authority remains honest during the permutation and decryption phases. Eperio's trust model requires honest commitment generation before the election and an honest bulletin board for posting results [21].

Usability

From the voter's perspective, Eperio is very similar to traditional paper ballots with one additional step to shred the candidate list after marking their ballot. This can introduce some complexity as the EA must instruct each voter to tear off and discard the candidate list. Otherwise, the voting experience is straightforward. Additionally, since each ballot is printed with a secret random ordering of candidates, the EA needs to handle a large set of unique ballots, which is a drawback noted in the literature. Eperio remains an academic proposal and has not seen large scale election use. Its security has been analyzed both probabilistically and through formal verification, with the latter identifying necessary corrections to prevent ballot stuffing and ensure ballot uniqueness [1, 21].

2.3.2 Scantegrity II

Scantegrity II is a **hybrid paper/optical-scan** voting system that adds cryptographic verifiability to traditional bubble ballots, introduced by Chaum et al. [22]. Each ballot is printed with invisible ink in each bubble that hides a unique confirmation code as shown in Figure 2.2 [4]. When a voter fills in a bubble (using a special pen that makes the invisible ink appear), a random code is revealed for that marked choice. The voter writes down this code on a detachable receipt and casts the ballot normally. After the election, all confirmation

codes (which are indexed by ballot ID and bubble position) are posted on a public bulletin board [23]. The voter can then verify *cast-as-intended* and *recorded-as-cast* by confirming that the codes on their receipt appear exactly once in the published list. If the voting machine were to alter the voter’s selection, the corresponding code would not match the published record, alerting the voter to a problem. Meanwhile, anyone can download the set of encrypted ballots and the associated proofs to independently rerun the tally process and the mix-net shuffling and decryption to verify the final count, achieving *counted-as-collected* verifiability.

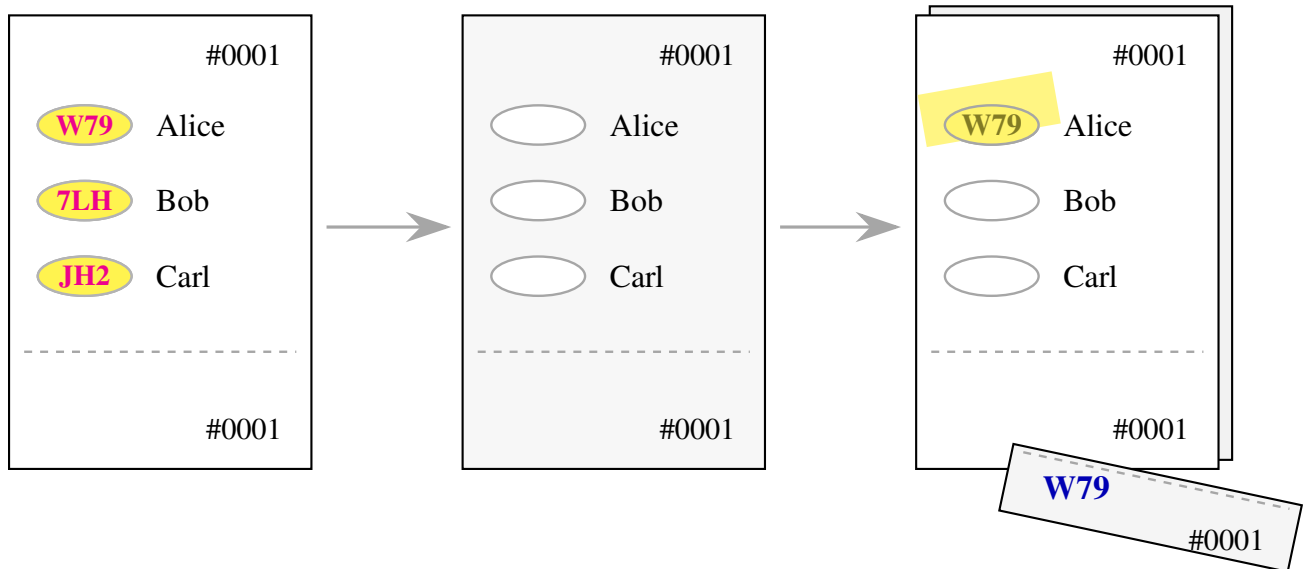


Figure 2.2: A Scantegrity II style ballot is shown with invisible regions indicated via a false-color overlay.

Cryptography and Verifiability

Scantegrity II uses a standard re-encryption *mix-net* to shuffle and anonymize the votes before tallying [23]. It then uses *non-interactive zero-knowledge proofs* for proving the mixing and decryption is done correctly. These proofs allow any observer to check that the published ciphertexts were properly shuffled and opened to produce the final tally, without revealing how any individual voted. The mix-net’s soundness relies on at least one EA being honest (a common assumption for mix-nets). For the voter, verification is straightforward

as the voter just needs to check their code on a website [22].

This protocol explicitly protects itself from **code-book**, and **randomization attacks** from adversaries. Since the confirmation code is a random value that is only revealed after the voter has marked their ballot, a coercer cannot ask that a voter produce a specific code in advance. In fact, a voter cannot even predict what code they will receive until they mark their ballot. This makes Scantegrity II *receipt-free* against common coercion threats, though full coercion-resistance requires stronger modeling and has been studied in depth in later work [24].

Privacy and Coercion Resistance

Ballot secrecy is also preserved in this protocol by the combination of secret codes and mix-net encryption. Even though the confirmation codes are published, no one can link a code to a particular voter or candidate. A voter could show their receipt to a third party, but because those codes are random and not known until the vote is cast, they do not provide a proof of which candidate was chosen. This prevents voters from creating a *predetermined receipt* for a coercer. A coerced voter cannot know in advance what code will appear, so they cannot satisfy the demands of a coercer for a specific code [25]. Thus, Scantegrity II provides a high level of *coercion resistance* short of *interactive* attacks.

Trust Model

Trust in Scantegrity II is distributed among the EA who run the mix-net and decryption process. It assumes at least one EA is honest so that the linkage between ballots and voters is concealed. The protocol also relies on the integrity of the printing process where the invisible ink codes printed on the ballots must correspond to the correct committed values used in the cryptographic tally. Additionally, the special pens used to reveal the codes must function correctly. There is no need to trust any voting machine software for protecting vote integrity, since even if a scanner does not record the marked positions correctly, the

code mismatches would expose it [3]. However, voters implicitly trust the codes revealed are in fact the ones associated with their candidate. This was addressed by conducting *print audits* of a subset of ballots pre-election [22].

Usability

Scantegrity II was highly praised for its usability. Voters mark a paper ballot in the usual way, and the only difference is writing down the revealed code on a receipt. The cryptography used in this protocol is largely transparent to voters and officials, and the system requires only minimal changes to existing optical-scan equipment (special pen and publishing codes).

Scantegrity II was piloted in Takoma Park, MD in 2009 with over 1,700 voters [3]. It was the first E2E-V protocol with ballot privacy used for a governmental election. Studies found that voters were able to use the system with minimal difficulty and understood how to check their receipts. However, only a subset of voters actually went online to verify their codes which shares a common challenge among protocols using public bulletin boards [26].

This does not undermine security if enough voters do check, but it shows a gap between system capability and user behavior. Scantegrity's limitations include the operational overhead of special pens and pre-printed ballots with invisible ink, and some added complexity for poll workers (they must manage the detachable receipts and ensure pens function properly). Nonetheless, Scantegrity II achieved full E2E verifiability for optical-scan elections, preserving traditional paper ballots, and demonstrated that usability and verifiability can coexist in a polling-place setting. It also set a standard for future systems by protecting against common vote-selling threats while remaining practical [22].

2.3.3 Civitas

Civitas is a fully remote, **internet voting** protocol and was one of the first systems to achieve provable coercion-resistance in an electronic voting context, introduced by Clarkson et al. [27]. It builds upon the JCJ (Juels, Catalano, Jakobsson) model for coercion-resistant voting [20]. In Civitas, each voter is issued a private voting credential, which is essentially a secret token, in addition to their identity. The protocol uses *threshold ElGamal encryption* for the votes and a mix-net to anonymize the encrypted ballots before tallying. Casting a vote requires the voter to encrypt their vote with the election *public key* and submit it along with a proof of validity tied to their credentials. The system publishes all the encrypted ballots on a public bulletin board. After the election, it also publishes proofs that each ballot is validly formed and included, and the decrypted tally with proofs.

Verifiability

Civitas provides both individual and universal verifiability, but in a different manner from the earlier paper-based schemes. For the voter, the *recorded-as-cast* verification is straightforward. The voter can check their encrypted ballot appears on the public bulletin board after submission, which confirms it was recorded. If the voter was coerced and submitted a fake vote as well, the fake one will also appear, but only the valid vote makes it through the tally. Civitas did not originally include a strong *cast-as-intended* mechanism. The voter is essentially trusting their computer to encrypt the ballot correctly. Researchers later suggested that a *Helios-style audit* or a *code-based check* could be added to Civitas to strengthen this aspect [28].

Civitas achieves universal verifiability by publishing the final decrypted tally along with zero-knowledge proofs that the tally corresponds to the set of posted ballots. Observers can check that all published encrypted votes are properly formed and that the threshold decryption of the sum corresponds to the announced results. In fact, anyone can verify that

each posted vote is valid by using plaintext-equivalence tests on the encrypted credentials, which ensure no invalid votes were counted without revealing any voter's choice, thereby achieving *counted-as-collected*. However, subsequent analysis identified vulnerabilities in the original plaintext-equivalence tests implementation that could allow authority collusion, highlighting the fragility of complex zero-knowledge proofs in practice [29].

Coercion Resistance

The main innovation of Civitas is its approach to providing strong coercion-resistance. If a voter is coerced into revealing their voting credential or is forced to vote under duress, Civitas allows the voter to invalidate the coerced vote without the coercer's knowledge. A coerced voter can intentionally *destroy* their true credential and cast a fake vote using a counterfeit credential. During the tallying phase, the mix-net and tallying officials will discard any ballots submitted with invalid or fake credentials. This is done through a plaintext-equivalence test that checks the validity of the voter's credentials, all without revealing the voter's identity. From the coercer's perspective, if they asked for the voter's credential, the voter can hand over a fake one and cast a corresponding fake ballot. The coercer sees a vote with the credential they know, but without them knowing, that ballot will be dropped and not counted. Meanwhile, the voter's real vote (cast with their real credential) remains private and will be tallied. Thus, the coercer cannot be sure if the vote they forced is the one that counted, achieving *coercion-resistance* by uncertainty. This technique is derived from the JCJ model and is provably secure under formal definitions of coercion-resistance [20].

Privacy

Civitas relies on a mix-net to separate voter identities from votes. Votes are posted alongside proofs of validity, but because of the encryption and mixing, the link between a voter and their vote is cryptographically hidden. Furthermore, fake credentials make sure even if

a coercer obtains a voter’s credential, they cannot definitively link that voter to a specific counted vote. Zero-knowledge proofs and plaintext equivalence tests are used so that ballot validity checks do not leak any information about the vote contents. This preserves *ballot secrecy* even against powerful adversaries. The threshold nature of the decryption prevents any single authority from decrypting the votes. Instead, a group of trustees must collude together, which adds robustness against insider threats [27].

Trust Model

Civitas requires a *trusted setup* for voter credentials, where a registration authority must issue each voter a secret credential via a secure channel. The security of the scheme relies on the assumption that the distribution is done honestly and that voters keep their credentials secret. The protocol also assumes a certain number of tally authorities are honest, which prevents all of them colluding to decrypt votes or to incorrectly count the votes. This is a *heavier* trust model than some other systems because it introduces a separate credential authority and multiple cryptographic steps that must be coordinated [30].

Although Civitas has been demonstrated in research settings, it has not been deployed in any large-scale governmental elections. Civitas provides strong theoretical security guarantees and E2E verifiability. However, the protocol introduces significant complexity, a less friendly voter experience, and a lack of a natural *cast-as-intended* audit mechanism. Basically, a voter has to trust their device to correctly form the ballot. Subsequent research has proposed integrating challenge mechanisms to address this gap in literature. Still, it remains a seminal work in the field of coercion-resistant internet voting, showing that strong privacy and verifiability can be achieved even in remote voting scenarios [30].

2.3.4 Helios

Helios is a pioneering **web-based** E2E-V voting protocol and one of the best-known electronic voting platforms in practice, introduced by Ben Adida [31]. It was designed for

low-coercion environments like university elections, board elections, and other civil society usecases rather than high-stakes political elections. Helios is implemented with a single election public key shared by trustees. It uses *homomorphic ElGamal public-key encryption* for votes, with a threshold decryption scheme for tallying [32]. Each voter logs into the Helios web portal, where they see a list of candidates and make their selection. The client-side JavaScript encrypts the vote in the browser before submission.

Helios offers an optional *audit* feature for *cast-as-intended* verification. A voter can opt to challenge the system by requesting their encrypted ballot be *open-audited* instead of cast. If the voter chooses to audit, the system reveals the plaintext of the *to-be* vote along with the randomness used, allowing the voter to verify that the encryption was correct for their intended selection. If the voter is satisfied with the audit, they can then fill out a fresh ballot and actually cast it. If the voter decides to cast without auditing, the encrypted vote is submitted directly to the public bulletin board [33]. In either case, the voter can later verify their encrypted ballot appears on the public bulletin board, thereby achieving *recorded-as-cast* verifiability. The optional ballot audit and the bulletin board check is meant to provide *cast-as-intended* assurance. As long as a sufficient amount of voters audit their ballots, a dishonest system cannot cheat without being detected with high probability [34].

Tally Verification

After the voting period ends, Helios allows anyone to verify the outcome by either homomorphically tallying the published ciphertexts or performing a shuffle, depending on the variant. In the homomorphic tally mode, all encrypted votes are multiplied together using *ElGamal addition* to obtain an encryption of the tally for each candidate. The trustees then perform a *threshold decryption* of this batched ciphertext to produce the final counts. Helios publishes *zero-knowledge proofs* at each step for proving each posted ballot is a valid encryption of some candidate, and the decryption of the tally is correct [35]. This satisfies the property of universal verifiability where any observer can check the tally corresponds to

the publicly posted ballots without trusting the authorities [36]. The entire process is transparent and auditable where the open-source Helios codebase allows independent auditors to replay and validate the election.

Privacy and Limitations

Helios provides *ballot secrecy* under the standard assumption that the threshold of trustees do not all collude. No single server learns the vote in plaintext, and without a full cooperation of all key-holders, or a break of the cryptography, individual votes remain encrypted. However, Helios does not provide strong *receipt-freeness* or *coercion-resistance*, as it trades those properties for simplicity. A voter receives an encrypted ballot as a receipt, and while that ciphertext alone does not reveal the vote, a coercer might attempt to get the voter's credentials or observe their voting session. Helios does not implement any mechanism to let voters safely lie to coercers, or to update their vote unlike Civitas [37]. The system's threat model assumes a low-coercion environment where voters are not selling votes or under duress, which is consistent with the protocol's usecases [31]. For *cast-as-intended* verifiability, if a voter does not perform the optional audit, they are implicitly trusting their device. This is a well known weakness where a compromised browser could encrypt a different vote than what's displayed. Unless the voter audits that ballot on another device, the manipulation would go undetected. The authors of Helios and subsequent researchers have acknowledged this risk and typically mitigate it by advising voters to audit at least one ballot [38].

Usability

The main advantage of Helios lies in its simplistic, yet accessible design. Voters can vote entirely from a web browser, without specialized hardware or physical ballots. This interface is similar to a standard online form, and except for the optional audit step, the voting process feels familiar (select and submit). Being easy to use, along with having its code open-source

has led to wide adoption in hundreds of binding elections in academia, student governments, professional societies, and other organizations. For example, Princeton’s undergraduate student government and the International Association for Cryptologic Research (IACR) have repeatedly used Helios for their elections [31, 39].

Individual verifiability is user-friendly, as voters receive tracking information and can simply look for their vote online. However, in practice, many voters do not bother to verify after casting. Universal verifiability is also guaranteed, provided by the public proofs. Yet, there have been cases where independent researchers detected issues (case of misconfiguration) by examining Helios election data [40]. This highlights the importance of *transparency* and *independent auditing* in E2E-V systems.

Helios has never been used in a high-stakes governmental public election. This is mainly because it lacks a mitigation for vote coercion, and because it produces no voter-verified paper record. Election authorities in governmental elections typically require a paper trail for auditing, while Helios is a fully digital system. Still, Helios remains a milestone in E2E voting, as it demonstrates secure online voting with verifiability is both feasible and practical [41]. Its weaknesses (notably around coercion) were considered acceptable trade-offs for its target usecases, where trust in the voting device is reasonable, and coercion is unlikely. Researchers have thoroughly analyzed Helios and while generally found it being *sound*, they identified subtle issues in early versions. This included certain side-channel attacks and improper use of randomness, which were addressed in later updates [42]. Overall, Helios offers strong integrity guarantees and a pragmatic level of security for low-coercion elections, with the clear caveat that it is not intended for adversarial, high-stakes elections.

2.3.5 STAR-Vote

STAR-Vote (Secure, Transparent, Auditable, and Reliable Voting System) is an innovative **hybrid** voting system proposal that combines electronic voting with a *paper trail* and *risk-limiting audits* (RLAs), introduced by Bell et al. [43]. It was designed for in-person voting to leverage cryptography without abandoning the security of paper ballots. In STAR-Vote, a voter uses a *ballot-marking device* (BMD) with a touch-screen to make selections. After the voter confirms their choices, the machine prints two records:

- **Human-readable paper ballot:** This is a summary of the voter’s selections, which the voter deposits into a physical ballot box.
- **Cryptographic receipt:** This is basically an encrypted form of the ballot, printed as a QR code on a separate slip.

The voter verifies the paper ballot’s plaintext to ensure it reflects their intent, which is the authoritative record for a manual recount or audit [44]. The cryptographic receipt corresponds to the same voter choices in encrypted form. The voter can either cast the ballot (in which case the encrypted vote is also stored and later posted to a public bulletin board), or spoil the ballot if something is wrong (spoiling triggers a re-vote and also serves a security function, explained below). After casting, the encrypted ballots and proofs from all machines are published for everyone to see, while the paper ballots are secured for auditing.

Cryptography and Verifiability

The protocol is based on the well-established **Benaloh-Cramer-Gennaro-Schoenmakers** approach to verifiable voting, which is the same family of techniques used in Helios [45]. A set of trustees generate a threshold public key for encryption using multiparty computation (MPC). Each voting terminal encrypts each vote under this key and also produces a commitment of the ciphertext using hashes as part of the receipt. The system ensures that

the QR code and hash on the voter's receipt match to the specified encrypted ballot on the bulletin board. This means if the machine tried to encrypt a different vote than the one printed, the codes would not match the published data. Voters thus achieve *cast-as-intended* by comparing their human-readable paper ballot to the cryptographic receipt, and ensuring the code appears on the public bulletin board associated with an encryption of the same choices. The voter does not decrypt the code but trusts that if the paper is correct and the code is posted, the encryption must correspond to that paper, since any mismatch would be detected by system audits. For *recorded-as-cast* verifiability, the voter can later check that their receipt's code is shown on the bulletin board. If the code is missing, the voter has evidence their ballot was not properly recorded.

STAR-Vote uses homomorphic tallying where all encrypted votes are summed up without decrypting individual ballots and then a threshold decryption of the final sum is performed. The trustees publish zero-knowledge proofs that this decryption is correct and corresponds to the published encryptions, which satisfies *counted-as-collected* verifiability. This provides mathematical proof that the electronic tally is faithful to the encrypted votes on the bulletin board. Additionally, STAR-Vote includes a *risk-limiting audit* of the paper ballots as another form of verification. This is where a random sample of the paper ballots is manually counted to ensure they statistically match the electronic results. This means even if the cryptographic part were somehow subverted, the paper trail would catch discrepancies with high probability [46].

Security

One of STAR-Vote's key security features is its protection against *compromised* voting machines. If a machine were malicious, it might try to print a valid paper ballot to fool the voter, but encrypt a different vote in the QR code to cheat electronically. STAR-Vote addresses this with a mechanism similar to Benaloh challenges. If a voter spoils their ballot, the machine is forced to reveal the encryption randomness and allow a full check of the

encrypted vote. Basically, a spoiled ballot can be taken as a *challenge*, where the voter or auditors can decrypt the specific ballot using cooperation from the authorities to confirm the machine's encryption was honest. Voters are expected to spoil a certain fraction of ballots randomly so that any widespread encryption fraud by a machine would be caught with high probability. Meanwhile, keeping the paper ballots separate and secure means that even if encryption went wrong undetected, the risk-limiting audit (RLA) provides a safety net. The RLA samples paper ballots and if the electronic tally does not match what the papers show for the sample, authorities can expand the audit or even do a full manual count. Essentially, STAR-Vote provides multiple verifiability measures including individual, universal, and conventional (paper recounts) verifiability [47].

Privacy

Votes are encrypted under threshold keys and shuffled homomorphically, so *ballot secrecy* is maintained as long as the trustees do not all collude to reconstruct individual votes. The use of a mix-net and homomorphic tallying means no individual encrypted ballot is decrypted alone, preventing the linkage of receipts to results. The paper ballot is anonymous, as it contains no identifying information about the voter and stays in a ballot box. Since voters do not take home the paper or any candidate identifying information, the system satisfies *receipt freeness* [48]. The receipt does not reveal the vote to a coercer as it is simply an encryption, which without the keys cannot be decoded. Additionally, since there is also the paper ballot backup, a voter under duress could agree to vote a certain way but then spoil the ballot without the coercer knowing the outcome (though this scenario depends on election procedures). In general, STAR-Vote preserves *ballot secrecy* comparable to standard encrypted systems and adds no new ways for coercion beyond what a paper ballot system might have. We note that paper systems themselves rely on procedures like private booths and prohibition of cameras to prevent coercion [43].

Trust Model

The protocol's design intentionally minimizes the need to trust any single component. Trust is distributed among multiple trustees for key management and decryption proofs, and the use of paper ballots provides a physical audit trail that does not depend on any software. Even if all voting machines were corrupt, as long as some voters spoil ballots and the paper audit is conducted, large-scale fraud would be detected [49]. STAR-Vote does not require trusting the software of the voting machine entirely, because any cheating done by the machines can be caught by either the cryptographic proofs or the paper audit, or both. However, the voter must trust the cryptographic tools used in the protocol are implemented correctly, and that the physical chain-of-custody for paper ballots is maintained. The innovation is that no single point of failure exists. An adversary would have to subvert the software, all trustees, and the audit process to undetectably cheat.

Usability

From the voter's perspective, STAR-Vote's experience is similar to using a modern touch-screen voting machine, but with the added benefit of a voter-verified paper trail [50]. The additional QR code does not require voter understanding as the voter only needs to verify the human-readable summary. If they trust the system, they do not need to do anything with the QR code at the polling place. They can later check online for the inclusion of their receipt, but they still cannot reconstruct their vote from the code. Thus, the verification burden falls mostly on auditors and observers who will check the proofs and conduct RLAs.

STAR-Vote was never built commercially and thus has not been used in a public election. It was a design study developed by academics and election officials around 2013-2015, but voting vendors did not implement it. This is mostly due to the complexity and cost of overhauling machines to integrate the cryptographic functions. Nonetheless, STAR-Vote is looked at as a state-of-the-art model for an in-person E2E-V system. Its concepts such

as adding a tracking code to paper ballots, and using RLAs as a fail-safe have influenced newer systems like ElectionGuard [50]. The advantages of this protocol lie in its strong transparency and layered verification. However, the trade-offs include increased complexity in the system (combining software, hardware, cryptography and audit procedures is non-trivial), and the need for new voting machine hardware or retrofits to print and handle the receipts [51]. Election administrators found it to be too complex at the time, which is a key reason it has not yet been deployed. Still, STAR-Vote remains a reference design showing how to achieve E2E integrity without sacrificing a voter-verifiable paper record.

2.3.6 ElectionGuard

ElectionGuard is a **cryptographic voting toolkit** intended to *retrofit* E2E verifiability onto existing voting systems. Rather than a full voting system, it provides libraries and specifications that voting machine vendors can use. ElectionGuard uses an *additive homomorphic encryption scheme*, which is an *ElGamal variant* over a multiplicative group, and supports addition of votes, along with a public election key and threshold secret key shared by trustees. In a typical ElectionGuard deployment, each voter at the polling place uses a voting machine as per usual. The voter makes selections on a touch-screen and gets a printed paper ballot, which is the main record for audits or recounts. Under the hood, the device also encrypts the voter's selections and prints a short tracking code on the ballot. The encrypted vote and its tracking code are uploaded to a public bulletin board after the election. The voter can use this code to later verify their ballot is included in the public records [52].

Verifiability

ElectionGuard ensures *cast-as-intended* and *recorded-as-cast* in a split way. *Cast-as-intended* is mainly provided by the paper ballot where the voter visually verifies the printed names on the paper match their intended selections. *Recorded-as-cast* is achieved through

the tracking code mechanism. After the election, the voter can look up the tracking code on the public website and confirm their encrypted vote is present and unaltered. The tracking code is typically a hash or a QR code that uniquely identifies the ballot encryption. If the code is found on the bulletin board, the voter knows that *some* encryption of their ballot was recorded. It is important to note the code itself does not reveal anything about the vote's contents as it is just an identifier, so posting it doesn't compromise *ballot secrecy*. If a malicious device tried to change the vote, it would have to either change the paper (which the voter would catch), or post an encryption under the wrong code (which would be caught by cross checks, since the code is derived from the ciphertext in a verifiable way) [53].

Once all encrypted ballots are posted, ElectionGuard performs a *homomorphic tally*. The trustees then decrypt the aggregated ciphertext to obtain the final tally. Because of homomorphism, individual ballots are never decrypted, preserving secrecy. ElectionGuard outputs the final tally along with zero-knowledge proofs that the decryption was done correctly and that the tally corresponds to the set of published encryptions. These proofs prove that each encrypted ballot was well-formed and the final sum was decrypted honestly. This gives anyone the ability to verify the election outcome without trusting the election officials, thus providing *counted-as-collected* verifiability [52].

Privacy

Ballot secrecy in ElectionGuard is strong as long as the encryption remains secure and no more than a threshold of trustees collude. Since only the batching of the votes is ever decrypted, individual votes remain secret. Additionally, the link between a voter's identity and their tracking code or encrypted ballot is not published. It is important to note the code is random and does not reveal the voter's identity. Voters do not receive any information that would let them prove how they voted, as they do not have the decryption key. Thus, *receipt freeness* is preserved in the sense that whatever information the voter goes home

with, they cannot convince someone of their vote. However, since ElectionGuard is often used in precinct voting, voters typically do not even take home a receipt as the code is on the ballot which they cast. If a voter wrote down their tracking code, it still does not show how they voted, only that their ballot is included in the tally. Therefore, the system does not introduce new coercion risks beyond a normal paper ballot system [54].

On the other hand, coercion-resistance is not explicitly argued. ElectionGuard does not prevent a coercer from simply watching the voter or demanding the voter fill out their ballot in a certain way. It "does not inherently solve coercion beyond basic receipt freeness". Some implementations might allow re-voting as a mitigation, but that is outside the core ElectionGuard framework [54].

Trust Model

ElectionGuard attempts to minimize changes to existing processes, so it leverages the existing trust in physical ballots and adds cryptographic trust in a set of trustees. A voter must trust that the paper ballot is the ultimate ground truth [55]. For this reason, a manual recount of ballots serves as the final authority should there be any dispute. The cryptographic part requires trusting that at least one of the trustees in the threshold key setup is honest. The system also assumes that the device prints the correct selections on paper, but since the voter checks said paper, this is a *voter verification* step rather than a blind trust. Another assumption is that voters or auditors will actually perform the verification steps where they check the tracking codes and examine the proofs. If no one checks the bulletin board or proofs, a malicious authority could potentially cheat without being caught. This is a generic assumption in all E2E-V protocols where public verification is only meaningful **if someone actually does it** [8].

Usability

One of ElectionGuard’s main goals is to keep the voter experience as identical as possible to current voting [56]. From the voter’s perspective, the workflow is the same as using a ballot-marking device that prints a paper ballot. The only addition is the opportunity to check the tracking code online after voting, which not all voters may do. If they do not do so, the system’s integrity still holds as long as *some* voters or auditors do such checks. Because ElectionGuard is a modular library, it has been integrated in pilot programs rather than full elections. The system has been included in a trial in the *2020 U.S. elections* (piloted in Fulton, Wisconsin, and demonstrations in some counties in Ohio, Utah) [55]. Parts of ElectionGuard were also tested in *Microsoft’s Defending Democracy program*, and by voting machine vendors in mock elections. ElectionGuard’s design was made to be vendor agnostic, which means it can work alongside existing voting machine software. Jurisdictions could add E2E verifiability without replacing all equipment.

The innovation of this approach lies in the fact that the cryptography is separate from the core voting machine UI [52]. This enables auditing layers to be added post hoc. However, ElectionGuard also presents some limitations. It requires a proper implementation with a given election system, where any bugs in integration could undermine its guarantees. Additionally, it inherits the usual challenges of E2E-V systems. This is where it depends on some fraction of voters or third-party auditors to actively verify the published data, and it does not address social engineering or coercion threats explicitly. However, ElectionGuard has brought significant attention as being a practical way to bring *verifiability* to existing voting systems without a complete overhaul, and its open-source nature allows for independent scrutiny and improvement over time.

2.3.7 Voatz

Voatz is a controversial **mobile voting application** that was used in a few U.S. pilot elections for overseas and military voters between 2018-2020 [57]. Unlike the academic protocols discussed above, Voatz is a proprietary system that claimed to offer security and immutability through the *blockchain*, but independent analyses found it lacked E2E verifiability and was severely insecure. Voatz's front-end is a smartphone app that allows voters to submit ballots electronically. It advertises features like *blockchain-backed receipts*, *biometrics for authentication*, and *E2E encryption* of ballots. However, a 2020 security analysis by Specter et al. (MIT) and the cybersecurity firm Trail of Bits found that it lacked E2E verifiability and contained severe, exploitable security vulnerabilities [58, 59].

Verifiability

Voatz provides no mechanism for voters to independently verify if the app had recorded their choices correctly. There is no public bulletin board or tracking number a voter can check. Voters have to trust the Voatz servers entirely, thereby failing to achieve *cast-as-intended* verifiability. Similarly, voters have no way to confirm their ballot is included unaltered in the tally as Voatz does not publish anonymized receipts or a public list of votes, thereby failing to achieve *recorded-as-cast* verifiability. Finally, none of the tallying is provably transparent as the backend is effectively a black box, thereby failing to achieve *counted-as-collected* verifiability. As the analysts from Trail of Bits put it, Voatz "provides no meaningful end-to-end verification" and offers neither individual nor universal verifiability in a formal sense [60].

Security Vulnerabilities

Another concern the researchers found was the Voatz app and infrastructure had multiple vulnerabilities. For example, an attacker with root access on the voter's phone could

intercept and alter the voter's choices before they are sent, and the Voatz app's checking mechanism could be easily evaded. The network protocol was also found to potentially leak information about the vote, and since the blockchain was run privately by Voatz's servers, it did not protect against a server-side attack altering votes [61]. Basically, Voatz's *blockchain* is a permissioned ledger with no public auditability. Only Voatz and its partners have nodes, so it functions like a centralized database.

The advantage of blockchain is that it is tamper-evident, but because outsiders cannot observe the chain directly, and the system lacks independent auditing channels, the supposed *publicly verifiable* blockchain functions as a *black box*. The MIT analysis concluded that there were "numerous vulnerabilities [that] allow different kinds of adversaries to alter, stop, or expose a user's vote" [58]. This included the possibility of some malware on the phone or a malicious Wi-Fi network that could change votes undetected, or that insiders could corrupt the blockchain records without voters or observers noticing [62].

Privacy

In theory, Voatz uses encryption to protect ballot content, but the MIT researchers found ways that an attacker could still learn a voter's choices. One example given was that with root access, an attacker could *bypass defenses* and learn the user's vote [61]. Thus, Voatz failed to guarantee *ballot secrecy* against realistic threats. As for *coercion-resistance* or *receipt freeness* verifiability, Voatz actually provided voters with a receipt of sorts (the app would indicate that the vote was recorded on the blockchain). However, because the process was not transparent, this receipt was not independently verifiable by a third party. A voter under coercion could potentially show their app screen or a confirmation message, but the coercer would have to trust Voatz's confirmation itself. Regardless, since the system itself was insecure, discussing coercion-resistance is almost pointless as one could argue it is *trivially receipt free* because even the voter cannot prove their vote to an outsider (as nothing publicly verifiable is produced). However, the lack of any mechanism to change or

invalidate a coerced vote, or to provide a fake proof means if an attacker compromised the device, the voter had no recourse.

Trust Model

Voatz expects users to completely trust the vendor's infrastructure, which in this case is the app, the cloud servers, and the private blockchain. In Voatz, if the company's servers said a vote was counted, the voter has to believe it without independent evidence. This is the antithesis of E2E verifiability, which aims to remove the need for such trust. The MIT researchers also noted Voatz's *lack of transparency*, and *closed-source* code as critical weaknesses. While Voatz claimed to conduct bug bounty programs and security audits, these were primarily *black box* tests until the Trail of Bits assessment [63]. Even after the identified critical vulnerabilities, Voatz disputed many findings and continued operations, raising concerns about the company's responsiveness to independent, security research [64].

Deployment

Voatz was used in a very limited way. For example, West Virginia's 2018 midterm elections allowed overseas military from certain counties to vote via Voatz, and some jurisdictions (parts of Utah in 2020, and a few small municipal pilots) experimented with it as well [65]. The usage was very narrow, where there were tens of voters in some cases. These pilots were met with *intense* scrutiny and criticism from the security community. After the public disclosure of Voatz's vulnerabilities and a DHS security audit, West Virginia and others backed away from the app. The takeaway is that Voatz is **not** an E2E-V protocol at all, despite its marketing suggestions. It is essentially a conventional internet voting system with an attached marketing buzzword, *blockchain*. Voatz shows us that simply using buzzwords like *blockchain* or *encryption* does not guarantee security or verifiability. Without open, independent verifiability and rigorous peer review, a voting system cannot

be trusted. Voatz has since become a cautionary tale, demonstrating that *security through obscurity* and *lack of transparency* are unacceptable in voting systems. Various computer and election security research communities have extensively criticized Voatz’s approach, with multiple independent analyses identifying critical vulnerabilities. The consensus among academic security researchers is that it fails to meet basic requirements for secure, verifiable elections.

2.3.8 Comparative Analysis

Each of the above protocols shows a different balance of cryptographic mechanisms, verifiability guarantees, and practical considerations. Schemes that are based on *mix-nets* vs. *homomorphic tally* differ in trust assumptions (mix-nets need at least one honest mix server, homomorphic tallies need trustees who will not leak keys). Coercion-resistant designs often come at the cost of complexity or delayed feedback, whereas schemes like Helios prioritize simplicity and direct verification but assume a low-coercion environment. Hybrid approaches (STAR-Vote, Scantegrity) show that combining paper with cryptography can yield robust systems, but this is often with increased logistical complexity. However, the biggest lesson from these comparisons is that **no single system is perfect**. Each has advantages and limitations, and real-world adoption has been slow and cautious. Even the most theoretically sound systems face practical challenges like user behavior, where many voters that do not check receipts can undermine the assurance of individual verifiability. Table 2.1 summarizes key properties across these systems and compares the protocols on core properties derived from the above analysis.

2.4 Gap and Problem Statement

Despite decades of research into E2E-V voting, current protocols have yet to fully bridge the gap between cryptographic theory and real-world election practice. E2E-V protocols

Table 2.1: Comparison of various E2E-V voting protocols across key properties.

Property	Eperio	Scantegrity II	Civitas	Helios	STAR-Vote	ElectionGuard	Voatz
Core Crypto Primitives	Symmetric-key commitments; threshold key; pre-printed randomized paper ballots	Re-encryption mix-net; invisible-ink confirmation codes; NIZK shuffle/decrypt proofs	Threshold ElGamal; private credentials; re-encryption mix-net; ZK proofs (incl. PETs)	Homomorphic ElGamal; commitments; NIZK proofs; optional ballot-audit	Threshold ElGamal (BCC-S); cryptographic receipts; RLAs + paper record	Additive homomorphic ElGamal; tracking codes; threshold keys; NIZK proofs; paper record	Permissioned blockchain; encryption/PKI (claimed); biometric login (proprietary)
Cast-as-Intended?	✓	✓	✗	✓ (audit-dependent)	✓	✓	✗
Recorded-as-Cast?	✓	✓	✓	✓	✓	✓	✗
Counted-as-Collected?	✓	✓	✓	✓	✓	✓	✗
Ballot Secrecy	Strong (random order + shredding)	Strong (codes reveal nothing)	Strong (mix + encryption; trustee threshold)	Strong (threshold decryption assumptions)	Strong (crypto + paper separation)	Strong (standard encryption + threshold)	Questionable (closed/proprietary; reported issues)
Receipt-Free?	✓	✓	✓	✗	✓	✓	✗
Coercion-Resistant?	✓	✓	✓	✗	✓	✗	✗
Universal Verif.?	✓	✓	✓	✓	✓	✓	✗
Voter Verif.?	✓	✓	✗ (partial)	✓ (audit-dependent)	✓	✓	✗
Trust Assumptions	At least 1 honest trustee; trusted ballot printing	At least 1 honest mix trustee; trusted printing/ink integrity	Trustees + credential system; client trust remains	Client + trustees; audit uptake affects assurance	Paper chain-of-custody; correct implementation; audits run	Paper printing/handling + crypto correctness; voters may check summaries	Trust vendor/device; limited public oversight
Usability	Paper + extra shredding step; non-standard ops	Mark normally; later code check; special pens/printing	High complexity (credentials + remote workflow)	Very usable web UI; audits often skipped	Polling-place friendly but adds scanning/spoil/audit steps	Fits polling place; minimal extra voter steps	Convenient mobile UX; weak security properties
Deployment	Academic prototype; limited public testing	Used in binding municipal elections	Research prototype; limited real deployments	Widely used for low-stakes elections	Design/pilots; not widely deployed	Pilots/demos; ecosystem adoption growing	Limited pilots; later stopped amid concerns
Formal Proofs?	✓	✓	✓	✗ (partial)	✗ (partial)	✓ (core crypto)	✗

such as Helios, Scantegrity II, Civitas, and Eperio all provide strong integrity guarantees in principle, but they suffer from important limitations that prevent them from being used in public elections. This section identifies key gaps in existing E2E-V voting protocols and defines the problem statement for the thesis.

- **Usability:**

Many E2E-V voting systems remain overly complex or insufficiently intuitive for the average voter and election official. For example, Civitas requires voters to manage their cryptographic credentials across multiple trustees and to perform verification steps that are not straightforward. Even in simpler systems, studies have shown that voters rarely engage with the verification procedures. Few voters actually check the printed audit trails or receipt codes when provided [31]. These limitations weaken the E2E goal. If the users are unable or unwilling to perform verification, the intended verifiability properties become largely theoretical.

- **Lack of Automated Verification:**

Current E2E-V implementations rely on ad hoc or voluntary verification rather than built-in, automatic checks. In Helios and similar *open audit* systems, it is on voters or observers to download the ballots and cryptographically verify the tally. As described earlier, this is often left undone. Similarly, Eperio's approach requires the EA to separately check proofs to ensure that *someone* actually looked over the published proofs. The recurring theme among such systems is that verification is an extra step, and without automated tooling, it may not happen at all. If no party takes the initiative to check the proofs or receipts, any potential fraud could go unnoticed. Because there is no automated verification, the integrity guarantees of such protocols may not be realized in real elections.

- **Deployment and Scalable Complexity:**

The cryptographic tools used in many E2E-V protocols are often complex and difficult to implement at scale. Protocols like Civitas and JCJ, which achieve very strong coercion resistance and verifiability, come at the cost of high computational overhead. These complications are significant barriers for real-world adoption. The EA must manage multiple trustees, cryptographic keys, and protocol steps that can degrade the performance for large elections. Even the protocols that have a more streamlined design require publishing or processing large proofs and ballot transcripts, which can be cumbersome. These hurdles in deployment make it challenging to integrate E2E-V protocols seamlessly into the constrained environment of real elections.

- **Reliance on Procedural Safeguards:**

One of the most significant gaps in literature is the trust model assumption. Most protocols assume that certain entities (election authorities, trustees) are either fully or partially trusted based on cryptographic guarantees. They replace cryptographic guarantees with procedural controls. Even traditional paper-based elections depend on chain-of-custody, random audits, and legal deterrents to ensure integrity. That

is why in most deployed E2E-V systems, critical guarantees, such as ballot privacy, remain contingent on *black box* assumptions and external procedural trust rather than cryptographic enforcement.

2.4.1 Realpolitik vs. Realcryptik: A Framework for Protocol Design

These shortcomings show the gap between cryptographic ideals and practical realities in elections. We can frame this trade-off in voting protocols as one between *Realpolitik* and a more deliberate philosophy, *Realcryptik*.

Realpolitik

Realpolitik is a political science term that prioritizes pragmatism and survival over strict adherence to ideological principles [66]. In Chapter 3, we explain the current scene of voting taking place in Canada, and more specifically in Ontario, and how the Realpolitik ideology is reflected across the election process. For E2E verifiability, this ideology translates to replacing cryptographic ideals (such as coercion resistance or ballot secrecy) for administrative feasibility and ease of use. Voting protocols that are designed under this philosophy accept these trade-offs that can have unavoidable consequences in real-world implementation. The rise of BMDs and the reliance on audits show the Realpolitik ethos. Election officials have chosen systems with known verification weaknesses in exchange for operational simplicity (discussed more in Chapter 3).

Realcryptik

On the other hand, we came up with the term *Realcryptik*, which represents an alternative to the Realpolitik compromise. We advocate for a *principled* pragmatism that stays true to cryptographic principles and keeps formal verification as a non-negotiable constraint. The Realcryptik approach accepts that some theoretical security ideals can be relaxed to

achieve deployability. However, this is only acceptable when the system can mathematically guarantee that the election’s integrity remains intact. The main distinction lies in *which* properties are relaxed and *how*. Realcryptik demands that any relaxations be explicit, justified, and bounded. Any constraints that are relaxed must be specified, and alternative enforceable guarantees must be enforced cryptographically, rather than relying on trust or legal safeguards alone.

We can frame existing protocols through a *Realcryptik* lens to see how they compare to this philosophy. For example, Scantegrity II can be seen as an early example of Realcryptik thinking. Rather than letting go of verifiability entirely, Chaum et al. made a practical choice to relax ballot secrecy against optical scanners, while still preserving E2E verifiability of the tally through cryptographic commitments. This is a *reasonable*, not an unprincipled compromise. The machine-side privacy loss is bounded and acknowledged, while the transparent tally verification is grounded.

We argue that Realcryptik should be an explicit design philosophy for E2E-V protocols. This design includes:

1. Identifying which security properties are non-negotiable (tally correctness) and which ones can be relaxed (ballot privacy against scanner machines).
2. Providing formal justification for each relaxation, grounded in realistic threat models and operational constraints.
3. Ensuring the *reduced* security model can still be audited and verified without intervention.

2.4.2 Zeeperio: A New Compromise for Automated Verifiability.

This thesis addresses the above gaps by exploring a new protocol design inspired by the Realcryptik approach. The proposed protocol, Zeeperio, builds on the Eperio system and

offers a unique compromise. It relaxes the assumption of strict ballot secrecy vis-à-vis the election authorities in order to achieve scalable, automatic verification, while still upholding voter privacy from the voters' perspectives (receipt freeness).

Zeeperio is designed to allow the EA to collect and handle ballots in open commitment form, much like current paper elections do, relying on legal and procedural safeguards to prevent any misuse of that information. By not hiding raw ballot selections from the EA, the protocol's threat model is simplified. This relaxation makes it possible to streamline the cryptography. Rather than performing a complicated encrypted tally or shuffling process, the system can publish a zero-knowledge proof that the recorded votes were tallied correctly. Zeeperio replaces Eperio's *full open* commitment auditing scheme with succinct zk-SNARK proofs that automate the verification of the election outcome. The result is that verification no longer depends on a dedicated human effort or procedural audit. It can now be performed automatically by an Ethereum smart contract in real time. This approach preserves receipt freeness (voters still cannot prove how they voted to a third party, since the public proofs and receipts reveal nothing that would violate the secret ballot) while vastly improving the accessibility and reliability of verification.

Chapter 3

3 Voting in Canada

3.1 Legal and Constitutional Framework

In order to evaluate the security of voting systems, we must define the parameters of correctness in terms of elections. In Canada, Section 3 of the *Canadian Charter of Rights and Freedoms* and the *Municipal Elections Act* provide the legal foundations for electoral processes [67, 68]. These legal frameworks offer guidelines and establish *ideal functionalities* that any voting protocol must follow to be considered valid.

3.1.1 Supreme Court - Public Confidence

The legitimacy of democratic elections in Canada is linked to the public's confidence in the election system. In the case of *Harper v. Canada (Attorney General)* 1 SCR 82, the Supreme Court of Canada implies that maintaining confidence in the election process is a "pressing and substantial objective" [69]. This extends beyond the accuracy of the tally. The judgment implies that if voters lack confidence in the system, they will be discouraged from participating.

Harper establishes the election protocol to be **universally verifiable**. Voters should be able to verify the results are correct [67]. This is in complete contrast to the current deployment of proprietary election technology. Without the ability to verify the counting mechanism, this plants a seed of doubt in the system and undermines the confidence in the election outcome.

In a *black box* model, the public has to trust the result because the accuracy is assured by a private vendor. This model of *security by authority* contradicts the Supreme Court [69]. If the vote counting process is hidden behind non-disclosure agreements, the public cannot

rationally hold confidence in the outcome. To enable public confidence, the system requires *scrutiny through transparency*, a property that is not available in closed-source, proprietary hardware and software. Therefore, the *black box model* is a technological and security risk [70].

3.1.2 Opitz v. Wrzesnewskyj

The *certainty* principle in Canadian election law requires the election outcome to reflect the cast ballots. This was brought in the case of *Opitz v. Wrzesnewskyj*, where a federal election result was contested [71]. The Supreme Court stated that ballots should only be voided if there are irregularities that are "serious and capable of undermining the integrity of the electoral process" [72].

3.1.3 The Municipal Elections Act of Ontario (MEA)

The *Municipal Elections Act of 1996* enabled the widespread adoption of online voting in Ontario with its ambiguously worded provisions [73]. Although the MEA serves as the principal legal framework for municipal elections, it does not cover specifics for digital voting systems. The legislation is highly detailed for paper-based voting, ballot handling, ballot box sealing, and the presence of scrutineers during the count. On the other hand, Section 42 of the Act allows municipal councils to use "alternative voting methods, such as voting by mail or by telephone, that do not require electors to attend at a voting place" [74]. This clause introduces some major concerns when deploying online voting systems without a verification or security framework.

The MEA does not establish a regulatory framework or security standards for digital systems. There are no provincial standards for:

- Source code verification
- Cryptographic key management

- Risk-limiting audits
- Penetration testing requirements

The MEA delegates the entire responsibility of election integrity to the individual municipal councils [75, 76]. These officials often lack the technical expertise to evaluate the security of online systems. Up to 414 municipal councils make independent decisions about election security, which creates a heterogeneous attack surface [77]. This causes vendors to compete on price and user experience rather than security. Without a unified standard, municipalities cannot check the verifiability features and vendors are deterred from developing them due to the associated costs.

3.1.4 Principles vs Reality

Since there are no explicit technical regulations, legal scholars have inferred general principles from the MEA. These principles are: *secrecy*, *fairness*, *accessibility*, *integrity*, *certainty* and *eligibility*, and they often conflict with current technological implementations [78, 79].

Ballot Secrecy ensures that no person should be able to tell how a voter has voted [80]. However, online voting systems involve transmitting vote choices over the internet to a central server. This causes several problems as shown in Section 3.3.

Additionally, the principle of Certainty requires auditability [81]. Current *black box* systems provide assurances from the vendors rather than physical proofs. The vendor acts as a *Trusted Third Party* in a scenario where trust should be minimized or distributed.

Finally, the principle of fairness requires consistent treatment of all voters [77]. For example, network outages can disproportionately affect specific voters. It forces emergency extensions to the election that creates inequities in voting opportunities.

3.2 From Observation to Blind Trust

The shift from paper ballots to electronic systems in Canada has led to a decline in observability. This section explains the shift across the three phases: the federal benchmark, the provincial hybrid, and the municipal black box.

3.2.1 Manually Counted Paper Ballots

Federal elections in Canada serve as the benchmark for transparency and verification. The protocol is as follows:

1. Voters mark paper ballots by hand.
2. The ballots are put into a sealed box.
3. After the polls close, the election officials count ballots manually in front of scrutineers.

This protocol is straightforward, observable, and structured to preserve a record of voter intent. In cryptographic terms, this system satisfies software independence [82]. Because the original paper ballots are retained for scrutiny and recounts, any error can be detected by comparing the results with the underlying ballots. The trust model is grounded in distributed observation as thousands of independent scrutineers monitor the ballot boxes which makes it infeasible to alter the results without detection [83]. This design is closely aligned with the constitution as it provides *electoral integrity* and *public confidence* [84].

In the adversarial model, we assume the individual poll workers may act dishonestly. However, their behaviour is localized and observable by scrutineers. Under this model, a successful large-scale attack would require coordinating many co-conspirators, making systemic fraud practically impossible [81].

3.2.2 Optical Scan Tabulators

Ontario introduced optical scan tabulators in the 2018 provincial elections [85]. This is the first step towards the *black box* paradigm. Voters mark a paper ballot which is fed into a scanner that tallies the votes digitally.

This results in a *hybrid* trust model. While the paper ballots are kept as a fail-safe, the ballots are counted by proprietary hardware and software [86]. This introduces some significant vulnerabilities:

- **Loss of Scrutiny:** Scrutineers observe the machine, not the ballot. They cannot verify the scanner counted the voter's choice correctly [81].
- **Centralized Vulnerability:** A faulty machine or malware injection in the software could affect results across the entire province [87].
- **Lack of Audits:** Despite relying on machines for counting, Ontario did not mandate routine risk-limiting audits (RLAs) to verify machine counts against paper trails. Without mandatory audits, the machine count is accepted by default. Unless there is a dispute post-election, the verifiability is limited to what can be shown electronically [88].

Although these vulnerabilities exist, the paper ballot trail is still retained so the system is technically recoverable. A full manual recount can restore the *Certainty* principle.

3.2.3 Municipal Online Voting Black Box

Municipal online voting eliminates the physical evidence entirely. As shown in ??, approximately 1.5 million voters were eligible to cast ballots electronically in the 2018 Ontario municipal elections [79, 89].

The election is conducted entirely on remote servers in jurisdictions that rely exclusively on

Category	Electronic Ballot	Electronic & Paper	Paper Ballot (Manual/Tabulator)
Municipalities	131	46	240
Percentage of Municipalities	33.5%	11.8%	54.7%
Eligible Voters	1,512,076	1,230,019	6,702,533
Percentage of Voters	16.0%	13.0%	71.0%

electronic voting. This marks the shift from a *Trust but Verify* model to the *Trust the Vendor* model. There is no physical ballot to recount. The ballot is essentially a database entry that can be altered by anyone with root access to the server. Additionally, the tally correctness depends entirely on the integrity of the proprietary code running on that server.

This represents a complete *Black Box* system where there is no *individual verifiability* or *universal verifiability* [90]. The voter cannot prove their vote was counted and the public cannot prove the final tally is correct. This model relies on the principle of *security by obscurity*, which is widely rejected in modern cryptography [91].

3.3 Black Box Security Analysis

Online voting has been adopted municipally without a corresponding security framework. This section analyzes the vulnerabilities of the current *Black Box* implementation.

3.3.1 2018 Bandwidth Crisis

Availability is an important security property in distributed systems that guarantees a system will continue to process transactions. In the voting context, this means the voters' ability to cast their ballots before the deadline. The 2018 Ontario municipal elections are a prime example of this shortcoming.

On election night (October 22, 2018), the vendor Dominion Voting Systems experienced

an availability failure [92]. There was a bandwidth limitation at a third-party data center causing throttled traffic. Voters in 43 municipalities were unable to login or experienced latency issues two hours before polls closed [93].

The denial-of-service (DoS) caused at least 35 municipalities to declare emergencies and extend voting for up to 24 hours [94]. While there was no lost data, this incident shows the fragility of a centralized infrastructure. Unlike a paper polling system where any disruption is limited to a specific location, a failure within a centralized server can impact voters across the entire network [95]. This is a violation of the *Fairness* principle, as extended voting hours may benefit certain candidates.

3.3.2 Side-Channel Attacks

The current implementation of online voting relies on Transport Layer Security (TLS) to encrypt the communication between the voter and the election server. While TLS protects the content of the vote, it does not necessarily protect the metadata or traffic patterns.

Research into side-channel attacks on web applications has shown one can use the size and timing of the encrypted packets to determine a user's choices [96]. In an election setting, voting for Candidate A might generate a different packet size and sequence than voting for Candidate B. Someone observing the network could potentially distinguish between the voter choices despite the TLS encryption [97]. Therefore, this is non-compliant with the legal principle of ballot secrecy.

3.3.3 Lack of Auditability

Auditability is essential for *public confidence*, and it is largely lacking in current municipal systems [81]. In cryptographic terms, the systems do not provide:

- **Individual Verifiability:** Voters do not receive a receipt proving their vote is included in the tally.

- **Universal Verifiability:** Observers cannot verify that the tally is summed up correctly.

Post-election surveys found that 22 municipalities in Ontario reported taking no independent security measures beyond what the vendor provided [77]. They treated the system as a *turnkey solution*. If a vendor's system was compromised, there would be no independent record to dispute the fraudulent results.

In 2005, Quebec experienced malfunctioning of their electronic voting machines. This led to a public outcry and a ban on municipal electronic voting province-wide, which has lasted over two decades [98]. This shows that a lack of verifiability can lead to public rejection of new technologies. Therefore, public confidence is an important requirement.

3.3.4 Authentication

Online voting systems often rely on a personal identification number (PIN) mailed to the voter, and the date of birth for authentication purposes. Unfortunately, this entropy is insufficient. In any given municipality, the probability of two voters having the same birthday is 100% (Birthday Paradox). A potential attacker can intercept a voter's PIN through the mail and easily brute-force their birthday or find it from public records [76, 99]. This weakness can lead to an attacker voting on behalf of unsuspecting citizens. The lack of proper authentication and identity verification undermines the integrity of the election.

3.4 A Review of Compliance

The failure to establish clear standards is a failure of governance. Under the current wording of the MEA, municipalities can choose systems based on cost and convenience rather than security [76]. Vendors often make unsupported claims regarding *military-grade encryption* or *blockchain-like security* without ever having their code publicly reviewed or third-party tested [80].

The online voting systems do not meet the international standards for e-voting (such as those proposed by the Council of Europe) in several areas [100, 101]:

- **Source Code Transparency:** Code is proprietary and closed-source.
- **Duty Separation:** Vendors often control the authentication, ballot casting, and tallying servers. This creates a single point of failure for potential manipulation.
- **Voter Anonymity:** In many systems, the voter's identity is linked to their ballot in the database to prevent double voting. This completely relies on software logic rather than cryptographic commitments to separate identity from vote choice.

The Digital Governance Council began drafting national standards for online voting in 2021 to address best practices for online voting. However, these remain voluntary [102, 103]. The province of Ontario has not mandated any of these standards which has left municipalities exposed. They end up relying on the vendors that do not hold any responsibility in the event of system failures.

3.5 Cryptographic Solution

By analyzing the current state of black box systems, there is a clear set of constraints that are not met. Legally, the system must comply with the principles of *certainty* and *public confidence* to be **Software Independent** [82, 104]. Since tabulators and online voting significantly improve accessibility and efficiency, we must adapt to this reality. We need to use modern cryptography in the digital space to restore transparency and verifiability in the voting infrastructure.

3.5.1 Bridging the Gap

A Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) protocol offers a solution to the election problem [105, 106]. In this protocol, a prover can

convince a verifier the tally is correct without revealing any information about individual votes. Zk-SNARKs satisfy the following properties:

- **Ballot Secrecy:** Zero-knowledge proofs ensure none of the ballot information is revealed.
- **Certainty:** Cryptographic proofs provide a higher level of certainty than vendor assurances.
- **Transparency:** The verification protocol is open-source and universally verifiable.

3.5.2 Zeeperio

The Zeeperio protocol addresses the gaps identified in the Canadian elections. It replaces the *black box* with a *glass box*, a system where users can fully verify the final tally without revealing any voter-specific data. By leveraging zk-SNARKs, Zeeperio offers the following:

- **Individual Verifiability:** Voters can check their encrypted ballot is on the public bulletin board.
- **Universal Verifiability:** Any observer can verify the proof to ensure the tally is correctly calculated from the encrypted ballots.
- **Coercion Resistance:** Each voter can verify their ballot was included in the tally without revealing which candidate they selected. The proof shows ballot inclusion, but does not leak the voter's choice. This way, a coercer cannot convincingly know how a voter actually voted. It is important to note this feature is missing in current online voting [107].

3.6 The Faith in Technology Bias

The push to implement unverifiable election systems is driven more by *faith in technology* rather than a commitment to security principles [80]. Municipal clerks are typically not equipped to evaluate cryptographic protocols and frequently mix the terms *user interface* for overall *system security*. The lack of any documented fraud is treated as proof of security, which is a logical fallacy that does not account for untested vulnerabilities. This bias causes vendors to focus on features and usability instead of security.

Additionally, this *faith in technology* bias is reinforced by the cost savings compared to using physical polls [108]. However, these savings come at the expense of electoral trust. As mentioned earlier in the Quebec case, when the trust is lost, it takes a huge toll to restore that trust.

3.7 Comparative Analysis of Canadian Voting Methods

We now analyse the security and legal criteria against the voting methods in Canada. Table 3.1 shows a concise overview of the trade-offs discussed in this chapter.

At the federal level, paper ballots and manual counting provide a highly distributed trust model in which altering the result would require compromising thousands of individuals. This provides a strong software independent system that aligns better with the legal requirement. Even though individual verifiability is limited, the strong universal verifiability principle strengthens public confidence in the voting process. Provincial optical scan tabulators introduce software dependence for tallying the results, but they retain a paper trail as a fallback measure. In this case, observers watch the machines instead of ballots, shifting the trust towards the assumption that the machines have been properly audited. On the other hand, municipal remote voting completely removes these core security properties. These systems depend on trusted third-party vendors, lack software independence, and expose

Criteria	Federal (Paper/Manual)	Provincial (Optical Scan)	Municipal (Remote Online)
Trust Model	Distributed	Hybrid	Centralized
Individual Verifiability	Low	Low	None
Universal Verifiability	High	Medium	None
Software Independence	Yes	Yes	No
Secrecy	Physical/Procedural	Physical/Procedural	Encryption (TLS) - Vulnerable
Availability Risks	Localized	Localized	Systemic
Legal Alignment	Strong	Strong (if audited)	Weak

Table 3.1: Security and Legal Analysis of Canadian Voting Methods

elections to significant availability risks. Moreover, they do not provide any individual or universal verifiability, leaving voters unable to fully confirm their ballots were counted and that the reported outcomes are accurate.

Chapter 4

4 Mathematical Foundations

This chapter introduces the mathematical concepts required to understand the Zeeperio protocol. It covers key topics such as finite fields, elliptic curves, commitment schemes (focusing on polynomial commitments via KZG), and zero knowledge proof systems (emphasizing on the PLONK protocol). We begin with basic field theory and gradually build up to the more advanced tools, ensuring the reader is fully equipped with all the necessary tools and notation.

4.1 Finite Fields and Polynomials

4.1.1 Finite Fields

A **Field** $(\mathbb{F}, +, \times)$ is an algebraic structure in which we can use arithmetic operations such as addition, subtraction, multiplication and division. If a field contains a finite number of elements, it is called a **Finite Field** [109]. We denote \mathbb{F}_q to be a finite field of order q ($|\mathbb{F}_q| = q$) where q is a power of a prime:

$$q = p^n$$

for some prime p and integer $n \geq 1$.

Every finite field satisfies an important property where the multiplicative subgroup of nonzero elements is cyclic of order $q - 1$. This means there exists a generator $g \in \mathbb{F}_q^\times$ such that every element can be written as g^k for some k [110].

4.1.2 Polynomials

Given a field \mathbb{F} , we can build a polynomial ring $\mathbb{F}[X]$ with polynomials

$$f(X) = a_0 + a_1X + a_2X^2 + \cdots + a_dX^d$$

where d is a positive integer and the coefficients $a_i \in \mathbb{F}$. We say $f(X)$ is a zero polynomial iff all the coefficients are zero and the result is always 0. For example, both $f(X) = 0$ and $g(X) = 0x^2 + 0x + 0$ are zero polynomials. We say the degree of a zero polynomial is:

$$\deg(0) = -\infty.$$

On the other hand, nonzero polynomials of degree d can have at most d roots in a field. This means there are at most d distinct $x \in \mathbb{F}$ such that $f(x) = 0$. If $f(x)$ has more than d roots, it must be the zero polynomial.

Vanishing Polynomials

If $\Omega = \omega_1, \omega_2, \dots, \omega_n \subset \mathbb{F}$ is a set of points, the vanishing polynomial over Ω is defined as

$$Z_\Omega(X) = \prod_{i=1}^n (X - \omega_i).$$

The vanishing polynomial $Z_\Omega(X)$ has a root at each $\omega_i \in \Omega$, and $\deg(Z_\Omega) = n$. Vanishing polynomials are useful for encoding boundary conditions, which we will use in our Zeeperio protocol [111].

Lagrange Basis

Polynomials are usually represented in **Coefficient Form** $\{1, X, X^2, \dots, X^d\}$. However, many protocols start with a vector of evaluation points requiring us to build a polynomial

that passes through them. This is where the Lagrange basis comes into play.

Given a domain $S = \{x_0, \dots, x_{n-1}\}$ and a set of evaluation points $\{y_0, \dots, y_{n-1}\}$, we want to find $P(X)$ such that $P(x_i) = y_i$.

We define the Lagrange basis polynomials $\mathcal{L}_i(X)$ for $i \in \{0, \dots, n-1\}$ such that:

$$\mathcal{L}_i(x_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Formally, the Lagrange basis is defined as:

$$\mathcal{L}_i(X) = \prod_{j \neq i} \frac{X - x_j}{x_i - x_j}$$

Here, $X - x_j$ ensures the polynomial is zero at every point in the domain except at x_i . We can then define the polynomial $P(X)$ as a linear combination of these basis elements:

$$P(X) = \sum_{i=0}^{n-1} y_i \cdot \mathcal{L}_i(X)$$

where $f(x_i) = y_i$ for $0 \leq i \leq n-1$ [112].

Schwartz-Zippel Lemma

Lemma 1. *Let $P(x) \in \mathbb{F}[X]$ be a nonzero univariate polynomial of degree at most d . Let $S \subseteq \mathbb{F}$ be any finite subset of the field. If r is chosen uniformly at random from S , then*

$$\Pr[P(r) = 0] \leq \frac{d}{|S|}.$$

The Schwartz-Zippel lemma tells us that two different polynomials of bounded degree are very unlikely to equal each other at a random point [113]. If a prover claims $P(X) = Q(X)$,

the verifier builds $R(X) = P(X) - Q(X)$ and evaluates it at a random point $\zeta \in \mathbb{F}$. If $P = Q$, then $R(X)$ is a zero polynomial. On the other hand, if $P \neq Q$, $R(X)$ is a non-zero polynomial of degree at most d .

The probability that the verifier accepts a false claim from the prover is $\frac{d}{p}$. For example, if $p = 2^{254}$ and $d = 2^{20}$, the probability of a false claim getting accepted is $\frac{1}{2^{234}}$, which is negligible. We rely on this lemma to argue soundness of polynomial protocols. Instead of checking $R(X)$ at every point on the domain, the verifier can check the relation at a single random point making the verification efficient.

4.2 Groups, Elliptic Curves, and Pairings

A group $(G, *)$ is a set G with a binary operation $*$ and the following properties:

- **Closed:** For any $a, b \in G$,

$$a * b \text{ is also in } G.$$

- **Associative:** For any $a, b, c \in G$,

$$(a * b) * c = a * (b * c).$$

- **Identity Element:** There exists an element $e \in G$ such that for every $a \in G$,

$$e * a = a * e = a.$$

In the multiplicative group, the identity element is 1 while in an additive group, the identity element is 0.

- **Inverse** For each $a \in G$, there exists $a^{-1} \in G$ such that

$$a * a^{-1} = a^{-1} * a = e.$$

Cyclic groups are generated by repeatedly applying this binary operation. If $G = \langle g \rangle$ is cyclic with generator g , then every element $h \in G$ can be expressed as $h = g^k$ using multiplicative notation, or $h = k \cdot g$ using additive notation for some k . The order of G is the smallest positive r such that g^r is the identity element. In many group-based protocols, we typically work in a subgroup G of a large prime order q , so the only subgroups of G are $\{e\}$ and G itself [114, 115].

4.2.1 Hardness assumptions

Many cryptographic protocols rely on certain problems being computationally infeasible for their security.

Definition 4 (Discrete Logarithm (DLog) Assumption). *Let $G = \langle g \rangle$ be a cyclic group of prime order q . The discrete logarithm problem is: given $h \in G$, find $x \in \mathbb{Z}_q$ such that $h = g^x$. The DLog assumption states that for all probabilistic polynomial-time (PPT) adversaries \mathcal{A} [116],*

$$\Pr[x \leftarrow \mathcal{A}(G, q, g, h) : h = g^x] \leq \text{negl}(\lambda),$$

where λ is a security parameter, $x \xleftarrow{\$} \mathbb{Z}_q$, $h \leftarrow g^x$, and the probability is over the randomness of x and \mathcal{A} .

Definition 5 (Elliptic Curve Discrete Logarithm (ECDLP) Assumption). *Let E/\mathbb{F}_p be an elliptic curve and let $G = \langle P \rangle \subseteq E(\mathbb{F}_p)$ be a cyclic subgroup of prime order q generated by a point P . The elliptic curve discrete logarithm problem is: given $Q \in G$, find $x \in \mathbb{Z}_q$ such*

that $Q = [x]P$. The ECDLP assumption states that for all PPT adversaries \mathcal{A} ,

$$\Pr[x \leftarrow \mathcal{A}(E, p, q, P, Q) : Q = [x]P] \leq \text{negl}(\lambda),$$

where $x \xleftarrow{\$} \mathbb{Z}_q$, $Q \leftarrow [x]P$, and the probability is over the randomness of x and \mathcal{A} [117].

Definition 6 (q-Strong Diffie-Hellman (q-SDH) Assumption). *Let G be a cyclic group of prime order q generated by g , and let $q_{\text{SDH}} \in \mathbb{N}$ be a parameter. Choose $x \xleftarrow{\$} \mathbb{Z}_q$ and give an adversary \mathcal{A} the tuple*

$$(g, g^x, g^{x^2}, \dots, g^{x^{q_{\text{SDH}}}}).$$

The goal of \mathcal{A} is to output a pair $(c, g^{1/(x+c)})$ for some $c \in \mathbb{Z}_q$ with $c \neq -x$. The q-SDH assumption states that for all probabilistic polynomial-time (PPT) adversaries \mathcal{A} ,

$$\Pr[(c, y) \leftarrow \mathcal{A}(G, q, g, g^x, \dots, g^{x^{q_{\text{SDH}}}}) : y = g^{1/(x+c)} \wedge c \neq -x] \leq \text{negl}(\lambda),$$

where λ is the security parameter and the probability is over the randomness of x and \mathcal{A} [118].

These assumptions formalize the idea that the group operation hides the exponents, providing one-wayness and pseudo-randomness that are crucial for cryptographic protocols.

4.2.2 Elliptic curve groups

An elliptic curve over a field \mathbb{F}_p is the set of solutions $(x, y) \in \mathbb{F}^2$ to an equation of form:

$$y^2 = x^3 + Ax + B,$$

where the identity element is the point at infinity O .

The curve forms an abelian group under addition law. If P and Q are two points on the curve, there exists another point where the line through P and Q will intersect. We can

evaluate this as

$$R = P + Q$$

after reflecting the point on the x-axis.

We denote $E(\mathbb{F})$ for the group of \mathbb{F} points on the curve. Typically, we work in a subgroup of $E(\mathbb{F})$ of a large prime order r where the curve has a small cofactor h such that $|E(\mathbb{F})| = h \cdot r$. The security of elliptic-curve cryptography is based on the elliptic curve discrete log problem (ECDLP).

4.2.3 Bilinear Pairings

We often require special pairing-friendly curves for certain protocols using bilinear pairings.

A pairing is a map

$$e : G_1 \times G_2 \rightarrow G_T,$$

where G_1, G_2, G_T are cyclic multiplicative subgroups of the same prime order r . G_1 and G_2 are usually distinct subgroups of an elliptic curve, and G_T is a subgroup of a finite field (often $\mathbb{F}_{p^k}^\times$ for some extension degree k). Pairings allow us to verify relationships between elements of G_1 and G_2 .

The pairing e has two main properties:

- **Bilinearity:** For any $a, b \in \mathbb{F}_r$ we have

$$e(aP, bQ) = e(P, Q)^{ab}.$$

This bilinearity means the pairing is a group homomorphism in each input coordinate.

- **Non-degeneracy:** $e(P, Q) = 1$ (the identity in G_T) for all $Q \in G_2$ if and only if P is the identity in G_1 (and vice versa). Equivalently, if P and Q are generators of G_1 and G_2 , then $e(P, Q)$ generates G_T . Thus the pairing does not send all pairs to the trivial

element, and in particular $e(g_1, g_2) \neq 1$ for generators g_1, g_2 .

Pairings are sometimes described as *encrypted multiplication*. They are used to check for multiplicative relationships between the exponents hidden in the group elements [119]. For example, if $P = a \cdot g_1$ and $Q = b \cdot g_2$, then $e(P, Q) = e(g_1, g_2)^{ab}$, revealing $a \cdot b$ in the exponent of G_T .

4.2.4 Algebraic Group Model (AGM)

When proving security of advanced protocols, we sometimes adopt the algebraic group model [120]. In the AGM, any adversary working in a group is assumed to be “algebraic,” meaning if it outputs any group element, it must also output a representation of that element as a linear combination of the group elements it has seen so far. In other words, the adversary cannot generate new group elements arbitrarily; it can only mix existing ones by known group operations. This is a weaker assumption than treating the group as a black box (the generic group model), but stronger than the standard model. The AGM has proven very useful in security proofs of cryptographic schemes that involve extractability or complex relations among group elements. For example, the security of the Groth16 SNARK (originally proved under a knowledge-of-exponent assumption) was reproved in the AGM by Fuchsbauer et al. under just the discrete logarithm assumption [120]. Intuitively, AGM forces any purported proof or commitment that an adversary produces to be derived from the legitimate powers in the setup, so if it manages to cheat, one can algebraically extract a solution to a hard problem (like DLog) from that output. We will assume the AGM when we discuss security proofs, as it underpins the knowledge soundness of our SNARK constructions in a relatively natural way.

4.3 Hash Functions and the Fiat-Shamir Transformation

4.3.1 Hash Functions

A *hash function* is a deterministic algorithm

$$H : \{0, 1\}^* \rightarrow \{0, 1\}^\kappa$$

that maps inputs of arbitrary length to fixed-length outputs of κ bits, called *digests*. They are designed as a one-way function with the following properties:

- **Collision Resistance:** It is infeasible for an adversary to find two distinct inputs $m, m' \in \{0, 1\}^*$ such that $H(m) = H(m')$.
- **Preimage Resistance:** Given a digest $y \in \{0, 1\}^\kappa$, it is infeasible for an adversary to find an input m such that $H(m) = y$.
- **Second Preimage Resistance:** Given an input m , it is infeasible for an adversary to find a different input m' such that $H(m) = H(m')$.

In many proof systems, these properties are important because it allows for the *derandomization* of the verifier's challenge. In an interactive protocol, the verifier samples a random field element to test the claimed relationship. However, in real-world scenarios, we often use non-interactive protocols because it eliminates the need for the back-and-forth communication between the prover and verifier. This also allows proofs to be generated and verified asynchronously. Therefore, the verifier's private randomness is replaced with publicly verifiable randomness derived from a hash of the protocol transcript.

4.3.2 Fiat-Shamir Heuristic

Suppose we have an interactive proof system where:

1. The prover sends a commitment α
2. The verifier responds with a random challenge c
3. The prover replies with a proof π
4. The verifier finally checks some relationship $\text{Check}(x, \alpha, c, \pi)$.

In a non-interactive proof system using the Fiat-Shamir transformation, the prover computes the challenge itself by hashing the transcript up to that point: $c = H(\text{pp} \parallel x \parallel \alpha)$, where pp denotes the public parameters and x is the public statement. The prover outputs the proof $\pi = (\alpha, c)$.

A verifier can deterministically recompute $c = H(\text{pp} \parallel x \parallel \alpha)$ and then check the proof by the same Check function. If the hash function H satisfies all the properties described earlier, the prover cannot predict or influence the outcome of c except by committing to α , and cannot change α afterward because it is part of the hash input. These properties carry over the *soundness* from interactivity to non-interactivity.

The Fiat-Shamir heuristic is applied to many protocols to eliminate interaction. We assume the random oracle model in our security arguments where we treat hash outputs as if they were truly random and uncontrollable by the prover.

4.4 Commitment Schemes

A commitment scheme allows one party (the committer) to *bind* to a value while keeping it hidden. Later, the committer reveals this value to a verifier. We can think of commitment schemes as a digital sealed envelope where one places a secret inside and later opens the envelope to show the revealed value matches the original claim. They are used in zero-knowledge proofs where a prover commits to hidden values that the verifier later checks without revealing the underlying data.

A commitment scheme involves two phases:

1. **Commit phase:** The committer chooses a message m and computes a commitment C :

$$C \leftarrow \text{Commit}(m)$$

which is sent to the verifier.

2. **Reveal (Open) phase:** The committer later reveals the message m and the verifier checks

$$\text{Commit}(m) \stackrel{?}{=} C.$$

Commitment schemes must satisfy two security properties:

- **Hiding:** It is computationally infeasible for an adversary \mathcal{A} to generate two messages $m_0, m_1 \in \mathbb{F}$ such that \mathcal{A} can differentiate between their corresponding commitments C_0, C_1 . Therefore, C reveals no information about m . Formally, for any probabilistic polynomial-time (PPT) $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we require:

$$\Pr \left[\begin{array}{l} (m_0, m_1) \leftarrow \mathcal{A}_1(1^k), \quad a \xleftarrow{\$} \{0, 1\}, \\ C \leftarrow \text{Commit}(m_a), \quad \tilde{a} \leftarrow \mathcal{A}_2(C) \end{array} \middle| \tilde{a} = a \right] \leq \frac{1}{2} + \text{negl}(k),$$

where $\text{negl}(k)$ denotes a negligible function in the security parameter k .

- **Binding.** It is computationally infeasible for an adversary \mathcal{A} to come up with a triple (a, b, b') such that (a, b) and (a, b') are valid commitments for m and m' and $m \neq m'$.

Formally, for any PPT \mathcal{A} we require:

$$\Pr \left[\begin{array}{l} (a, b, b') \leftarrow \mathcal{A}(1^k), \\ m \leftarrow \text{Open}(a, b), \quad m' \leftarrow \text{Open}(a, b') \end{array} \middle| m \neq m' \wedge m, m' \neq \perp \right] \leq \text{negl}(k),$$

where $\text{negl}(k)$ denotes a negligible function in the security parameter k .

4.4.1 Pedersen Commitment Scheme

A *commitment scheme* allows a prover (committer) to *bind* themselves to a value while keeping it *hidden* until they later choose to open it. Pedersen commitments are a standard, pairing-free commitment scheme based on the discrete logarithm assumption in a prime-order group. Unlike polynomial commitments, Pedersen commitments are primarily used to commit to *single* field elements (or vectors of elements) with strong privacy guarantees and simple verification [116, 121].

- **Setup:** Let \mathbb{G} be a cyclic group of prime order q written multiplicatively, with generator $g \in \mathbb{G}$. Choose an additional generator $h \in \mathbb{G}$ such that the discrete-log relation between them is unknown, i.e., no party knows $\alpha \in \mathbb{Z}_q$ with $h = g^\alpha$.¹ Publish public parameters $\text{pp} = (\mathbb{G}, q, g, h)$.
- **Commit:** To commit to a message $m \in \mathbb{Z}_q$, sample randomness $r \xleftarrow{\$} \mathbb{Z}_q$ uniformly and compute

$$C = \text{Com}(m; r) := g^m h^r \in \mathbb{G}.$$

The commitment C is a single group element. Intuitively, g^m binds the message, while h^r statistically masks it.

- **Open (Reveal):** To open C , the committer reveals (m, r) . This is the decommitment information (witness) demonstrating that C was formed honestly.
- **Verify:** Given (C, m, r) , the verifier checks

$$C \stackrel{?}{=} g^m h^r.$$

¹In practice, h is often derived from a domain-separated hash-to-group procedure to avoid any trapdoor.

If the equality holds in \mathbb{G} , the opening is accepted.

- **Security properties:**

- *Perfect hiding:* For any fixed $m \in \mathbb{Z}_q$, the distribution of $C = g^m h^r$ over uniform r is uniform over \mathbb{G} (up to a fixed shift), hence C leaks no information about m information-theoretically [121].
- *Computational binding:* Under the discrete logarithm assumption in \mathbb{G} and assuming the relation between g and h is unknown, it is computationally infeasible to find two distinct openings $(m, r) \neq (m', r')$ such that

$$g^m h^r = g^{m'} h^{r'}.$$

Indeed, such a collision implies a discrete-log relation between g and h .

- *Additive homomorphism:* Pedersen commitments are additively homomorphic:

$$\text{Com}(m_1; r_1) \cdot \text{Com}(m_2; r_2) = \text{Com}(m_1 + m_2 \bmod q; r_1 + r_2 \bmod q).$$

This property is frequently used to aggregate commitments and to prove linear relations in zero-knowledge [116].

4.4.2 KZG Commitment Scheme

A polynomial commitment enables a prover to convince a verifier that $f(x)$ has a value y at some point $x = z$. This is all done without revealing anything else about the polynomial f beyond the evaluation. While Pedersen commitments are efficient for committing to single values or vectors, many protocols require commitments to entire polynomials. In zk-SNARK protocols, the prover works with low-degree polynomials encoding various constraints. The verifier needs proof that the claimed evaluations are consistent with the

committed polynomial.

One of the main polynomial commitment schemes is the KZG commitment scheme, named after Kate, Zaverucha, and Goldberg [122]. KZG is one of the schemes requiring pairing-friendly curves as it uses elliptic curve groups and bilinear pairings. The commitment scheme is as follows:

- **Setup:** KZG requires a one-time trusted setup that generates a structured reference string (SRS) tied to a secret toxic waste $\tau \in \mathbb{F}_q$, where q is a large prime. Because the setup relies on a secret τ , it must be generated in a trusted way using multi-party computation as shown in Figure 4.1. If an adversary somehow learns the secret τ , they could produce fake proofs or break the binding of commitments without detection.

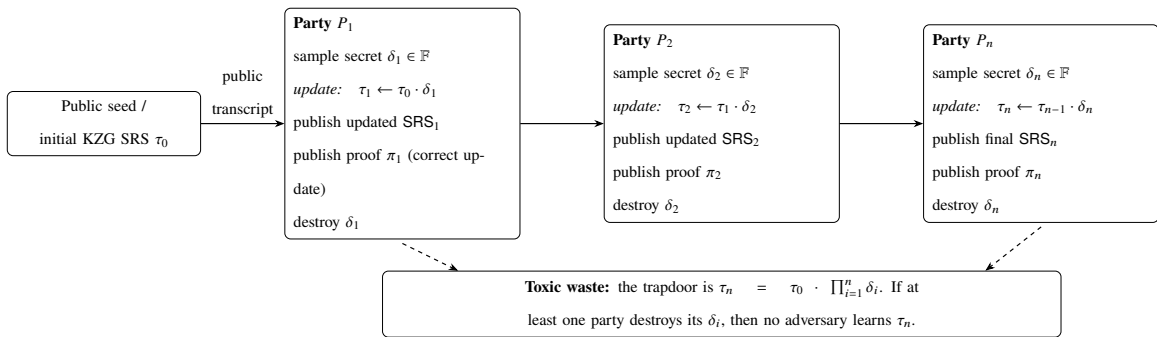


Figure 4.1: Multi-party trusted setup (MPC) for KZG.

The resulting SRS is built using powers of τ in the form of group elements:

$$\{g^{\tau^0}, g^{\tau^1}, \dots, g^{\tau^d}\} \in G_1 \text{ and } \{h^{\tau^0}, h^{\tau^1}, \dots, h^{\tau^d}\} \in G_2$$

where $G_1 = \langle g \rangle$ and $G_2 = \langle h \rangle$. The SRS allows committing to polynomials up to degree d . Once the SRS is published, the secret τ is discarded.

- **Commit:** For a polynomial $f(x)$ of degree $\leq d$ with coefficients (a_0, a_1, \dots, a_d) in

\mathbb{F}_p , the prover computes the commitment

$$C \leftarrow g^{f(\tau)} = g^{a_0 + a_1\tau + \dots + a_d\tau^d}$$

using the SRS. The commitment C is a single group element on the curve that binds to the polynomial f . After sending C to the verifier, the prover cannot change which polynomial they have committed to as any attempt to open C to a different polynomial will fail the verification checks.

- **Reveal:** Suppose the verifier wants proof that the committed polynomial f evaluates to y at some random point $z \in \mathbb{F}_q$. One way to prove the evaluation is by having the prover *reveal* the polynomial f , and the verifier can evaluate $f(z)$ itself. However, this is inefficient, and if the polynomial f encodes some private data, it loses all secrecy. Instead, the prover produces a *succinct* proof π for the claimed evaluation. The prover starts by computing the polynomial quotient

$$Q(x) = \frac{f(x) - y}{x - z}.$$

$Q(X)$ exists iff $f(z) = y$, in which case $x - z$ divides $f(x) - y$. Otherwise, $Q(X)$ would be a rational function to which we cannot commit to using the KZG scheme.

The prover then computes the evaluation proof π :

$$\pi = g^{Q(\tau)}$$

This evaluation proof can be thought of as the *witness polynomial* to the fact $f(z) = y$.

- **Verification:** The verifier checks the following pairing equation using the elements of the SRS:

$$e(C - g^y, h) \stackrel{?}{=} e(\pi, h^{\tau-z}).$$

All of the elements are either taken from the SRS or provided from the prover. Because of bilinear pairings, the verifier checks $f(\tau) - y \stackrel{?}{=} Q(\tau) \cdot (\tau - z)$. The pairing holds iff $f(z) = y$, which confirms the evaluation claim. The verifier only needs to do a constant number of pairing and multiplication operations regardless of the degree of f . Therefore, the proof is succinct and quick to verify.

KZG commitments are also homomorphic. If C_f and C_g are commitments to polynomials f and g , we can compute the following:

$$C_f \cdot C_g \leftarrow \text{Commit}(f + g)$$

This is useful for batching multiple proofs into one proof using *random linear combination*.

The security of the KZG scheme relies on the trapdoor secret τ being computationally infeasible to recover from the SRS. If τ were ever revealed or maliciously constructed, security breaks as a prover could fake proofs by exploiting knowledge of τ . Assuming the setup was done honestly, KZG commitments are secure and binding because finding $f(x)$ from C or forging a proof would require solving the q-SDH assumption (Section).

4.5 SNARKs

4.5.1 Interactive proofs and arguments

An *interactive proof system* is a protocol where a prover \mathcal{P} wants to convince a verifier \mathcal{V} of the validity of an NP statement x . After the prover sends all relevant proofs, the verifier outputs either accept or reject. Such protocols have the following properties:

- **Completeness:** If the claimed statement is correct and the protocol is followed honestly, the verifier accepts with overwhelming probability.

Definition 7 (Completeness). *Let R be an NP relation and let $\Pi = (\mathcal{P}, \mathcal{V})$ be an interactive protocol for R . We say that Π is complete if for every $(x, w) \in R$,*

$$\Pr [\text{Output}_{\mathcal{V}}(\Pi) = 1] \geq 1 - \text{negl}(\lambda),$$

where λ is the security parameter.

- **Soundness:** An interactive proof system $(\mathcal{P}, \mathcal{V})$ is *sound* if no cheating prover can convince the verifier to accept a false statement except with negligible probability.

Definition 8 (Soundness). *Formally, for every PPT adversarial prover \mathcal{P}^* and every statement $x \notin L(R)$,*

$$\Pr [\text{Output}_{\mathcal{V}}\langle \mathcal{P}^*(x), \mathcal{V}(x) \rangle = 1] \leq \text{negl}(\lambda).$$

where $L(R) = \{x : \exists w \text{ s.t. } (x, w) \in R\}$.

In *interactive proofs*, soundness is information-theoretic where the above bound holds even for an unbounded \mathcal{P}^* . In *interactive arguments*, soundness is computational where the bound is guaranteed only for a PPT \mathcal{P}^* under a hardness assumption.

In SNARK protocols, we require *knowledge soundness*. Any prover that convinces the verifier must *know* a corresponding witness.

Definition 9 (Knowledge Soundness). *Formally, $(\mathcal{P}, \mathcal{V})$ is an argument of knowledge for relation R if there exists a PPT extractor \mathcal{E} such that for every PPT adversarial prover \mathcal{P}^* and every statement x ,*

$$\Pr [\text{Output}_{\mathcal{V}}\langle \mathcal{P}^*(x), \mathcal{V}(x) \rangle = 1 \wedge (x, w) \notin R] \leq \text{negl}(\lambda),$$

where $w \leftarrow \mathcal{E}^{\mathcal{P}^*}(x)$ is the witness extracted by \mathcal{E} using oracle access to \mathcal{P}^* . Whenever \mathcal{P}^* convinces \mathcal{V} to accept x with non-negligible probability, the extractor can recover

a witness w such that $(x, w) \in R$.

- **Zero-Knowledge:** An interactive proof (or argument) $(\mathcal{P}, \mathcal{V})$ for relation R is *zero-knowledge* if the verifier learns nothing beyond the validity of the statement.

Definition 10 (Zero-Knowledge). *Formally, zero-knowledge is defined by the existence of a simulator that can generate transcripts indistinguishable from those of a real interaction, without knowing the witness. Let \mathcal{V}^* be any adversarial PPT verifier. The protocol is computational zero-knowledge if there exists a PPT simulator $\text{Sim}^{\mathcal{V}^*}$ such that for every x and witness w for $(x, w) \in R$,*

$$\text{Sim}^{\mathcal{V}^*}(x) \approx_c \text{view}_{\mathcal{V}^*}(\mathcal{P}(x, w), \mathcal{V}^*(x)),$$

where $\text{view}_{\mathcal{V}^*}$ denotes the verifier's view, and \approx_c denotes computational indistinguishability. If the two distributions are statistically indistinguishable, then the $(\mathcal{P}, \mathcal{V})$ for relation R is statistical zero-knowledge [123].

4.5.2 Polynomial Interactive Oracle Proofs (Poly-IOPs)

A Poly-IOP is an interactive protocol where the prover provides oracle access to polynomials $P_1(X), \dots, P_t(X)$ and the verifier queries these polynomials at randomly chosen field points. Instead of sending the polynomials directly, the prover *commits* to the polynomials and later responds with evaluation queries where *soundness* is provided by the Schwartz-Zippel lemma.

This setup is used for polynomial-IOP-based systems such as Sonic, Marlin, and PLONK. In PLONK-style arithmetizations, the prover represents witness assignments by polynomials $(w_1(X), w_2(X), w_3(X))$ over an evaluation domain H , encodes gate logic using selector polynomials, and enforces wiring constraints via a permutation (grand-product) argument with an auxiliary polynomial (often denoted $Z(X)$). The verifier samples random challenges

(often denoted $\beta, \gamma, \alpha, \dots$), and the prover aggregates many constraints into a small set of polynomial identities (typically through a quotient polynomial that should vanish on H if and only if all constraints are satisfied). The verifier then checks these identities at a single random point ζ , obtaining high confidence in global correctness from a small number of evaluations.

4.5.3 SNARKs: Succinct Non-Interactive Arguments of Knowledge

A SNARK is a proof system with

1. **Succinctness:** Proofs are polylogarithmic or constant size and verification is fast.
2. **Non-Interactivity:** Using the Fiat-Shamir heuristic, the prover only sends a single message.
3. **Computational Soundness:** It is an *argument* relying on hardness assumptions.
4. **Knowledge Soundness:** If a proof is accepted by a verifier, an extractor can recover a witness from any prover that produces such proofs.

SNARKs compile an interactive proof system or a Poly-IOP into a single message using commitments and sampling random challenges. A Poly-IOP is made non-interactive and succinct by combining polynomial commitment schemes with the Fiat-Shamir heuristic. In a Poly-IOP, the prover provides oracle access to a polynomial $P_i(X)$ whereas in SNARK systems, the prover sends a commitment $\text{Commit}(P_i)$. Whenever the verifier queries P_i at a point z , the prover returns $P_i(z)$ with an opening proof that *proves* the commitment corresponds to a polynomial evaluating to $P_i(z)$ at z .

By replacing the verifier's random challenges with the Fiat-Shamir heuristic, we can reduce the interaction between the prover and verifier into a single proof. The verifier recomputes these challenges, checks all the polynomial openings, and verifies the polynomial identities hold at the sampled points.

SNARK families

SNARK constructions are often organized into two broad families that illustrate the above compilation idea.

Pairing-based (QAP) SNARKs: Groth16. A seminal construction is Groth16 [5], a pairing-based SNARK for arithmetic circuit satisfiability. It arithmetizes a circuit into a *Quadratic Arithmetic Program* (QAP), yielding polynomials $A(X), B(X), C(X)$ and a vanishing polynomial $Z_H(X)$ such that satisfiable instances admit a polynomial $H(X)$ with

$$A(X) \cdot B(X) - C(X) = H(X) \cdot Z_H(X).$$

Groth16 uses a circuit-specific structured reference string (SRS) containing encodings of secret trapdoor values (commonly denoted $\tau, \alpha, \beta, \dots$) to produce extremely small proofs (three group elements) and very fast verification (a constant number of pairings, plus work linear in the public input size). The trade-off is that Groth16 requires a per-circuit trusted setup, since the SRS is tied to the QAP instance.

Polynomial commitment SNARKs: PLONK and related systems. PLONK [124] exemplifies the Poly-IOP approach: it reduces circuit satisfiability to checking a small number of polynomial identities over an evaluation domain, including both gate constraints (enforced via selector polynomials) and wiring constraints (enforced via a permutation/grand-product argument). The prover commits to the relevant witness and auxiliary polynomials, derives challenges via Fiat-Shamir, and opens a small set of polynomials at a random point ζ with evaluation proofs. With a constant-size polynomial commitment scheme such as KZG, the resulting proof and verification remain compact and efficient. A key advantage of PLONK-style systems is *universality* where a single updatable SRS can support many circuits up to a size bound, avoiding per-circuit setup. Sonic and Marlin similarly follow the "IOP + commitments + Fiat-Shamir" setup. They differ in arithmetization and optimization choices, and represent a major line of practical SNARK design.

Chapter 5

5 Zeeperio: Verifying Elections using Poly-IOP Gadgets and On-Chain Verification

5.1 Notation and Preliminaries

We consider an election with \mathcal{B} ballots and C candidates. There are $\mathcal{P} = \mathcal{B} \cdot C$ unique ballot-candidate positions listed in canonical order as shown in Fig. 5.1. Let $j \in \{0, \dots, \mathcal{B} - 1\}$ denote the ballot index and $k \in \{0, \dots, C - 1\}$ the candidate index within a ballot. Then $i = j \cdot C + k \in \{0, \dots, \mathcal{P} - 1\}$ iterates over each ballot-candidate position. Each position \mathcal{P} is associated with the following:

- **Ballot Column B_{ID}** : identifies each unique ballot \mathcal{B}_j listed in canonical order.
- **Confirmation Code Column $C_{confirm}$** : a unique confirmation code issued to each position \mathcal{P} . The voter is given the code corresponding to their vote.
- **Mark Column M** : indicates if position \mathcal{P} was voted for. $M \in \{0, 1\}$ where

$$M = \begin{cases} 1, & \text{if the ballot-candidate pair was marked} \\ 0, & \text{if it was unmarked} \end{cases}$$

- **Audit Column A** : indicates the audit status of ballot \mathcal{B}_j . $A \in \{0, 1\}$ where

$$A = \begin{cases} 1, & \text{for every } k \text{ in } \mathcal{B}_j \text{ if it was selected for audit} \\ 0, & \text{otherwise} \end{cases}$$

Figure 5.1: Example election data representation with three candidates

Ballot ID $B_{ID}[i]$	Audit $A[i]$	Mark $M[i]$	Confirmation Code $C_{confirm}[i]$
001	0	0	9263928202
001	0	1	4923486522
001	0	0	3976458823
002	1	0	1935498254
002	1	0	6482564825
002	1	0	1983252475
003	0	1	5249630250
003	0	0	3125019852
003	0	0	4630258701
...

For efficient FFT-based polynomial transformations, we require an evaluation domain size that is a power of two [125]. We take the \mathcal{P} real positions and add *padding bits* to build columns of length $n = 2^t$ for $t = \lceil \log_2 \mathcal{P} \rceil$. This results in i iterating over $\{0, \dots, n - 1\}$ where

$$n = \mathcal{P} + \underbrace{(n - \mathcal{P})}_{\text{padding}}.$$

All padded rows are set to zero ($M[i] = A[i] = 0$ for $\mathcal{P} \leq i < n$). We note the columns M and $C_{confirm}$ are private witness data and are never published.

5.1.1 Polynomial IOPs and KZG Commitments

Zeeperio is built using the *Polynomial Interactive Oracle Proof* (Poly-IOP) framework, which is the foundation for many SNARK systems such as PLONK, Sonic, and Marlin. In a Poly-IOP, the prover encodes each of the protocol’s constraints as a low-degree polynomial over a finite field. After the prover commits to these polynomials, the verifier queries

random evaluation points using an oracle. The prover opens the evaluations at the specified random points and sends the evaluations to the verifier. Finally, the verifier checks if the given evaluations satisfy the polynomial constraints [126].

Zeeperio uses the Poly-IOP framework by encoding the election columns as a set of polynomial identities. We then use algebraic operations to verify each constraint using the KZG polynomial commitment scheme to complete the full SNARK system. We denote the following operations in KZG:

KZG.Commit : $G_1 \leftarrow \mathbb{F}_{q[X]}_{\deg < d}$

For a polynomial $F(X) \in \mathbb{F}_q[X]_{\deg < d}$, the prover uses the SRS to compute a group element $C = \text{KZG.Commit}(F) \in G_1$ that cryptographically binds to F .

KZG.Open : $\pi \leftarrow (F, z)$

Given the polynomial $F(X)$ and an evaluation point $z \in \mathbb{F}_q$, the prover calculates $y = F(z)$ and an opening proof $\pi = \text{KZG.Open}(F, z)$ that proves y is indeed the evaluation of the committed polynomial at point z .

KZG.Verify : $\{\text{accept}, \text{reject}\} \leftarrow (C, z, y, \pi)$

Given the commitment C , an evaluation point z , the claimed value y , and the opening proof π , the verifier checks if C opens to y at z and outputs *accept* or *reject*.

We choose the KZG commitment scheme for several reasons. First, the commitments and evaluation proofs are *constant-size*, which is necessary when verifying on-chain using Ethereum smart contracts. Secondly, KZG has the following homomorphic property in the multiplicative subgroup:

$$\text{KZG.Commit}(f + g) = \text{KZG.Commit}(f) \cdot \text{KZG.Commit}(g).$$

This allows the prover to batch multiple polynomial evaluations into a single commitment for the verifier to check with a single pairing. Finally, since we design Zeeperio's constraints

explicitly as polynomial identities, we can directly use the succinctness and homomorphic properties of KZG to efficiently verify the constraints. Although Zeeperio is built using KZG commitments, the protocol can be generalized to support other commitment schemes as well.

As shown in Fig. 5.2, there are several ways of building a SNARK system. SNARKs are typically implemented at a higher abstraction level with a *circuit*, or a *program* running inside a zkVM. The program is first converted into a circuit representation. The circuit is then compiled into a constraint system represented by Poly-IOP gadgets. These gadgets consist of selector polynomials, wire polynomials, permutation arguments, and grand-product checks [127].

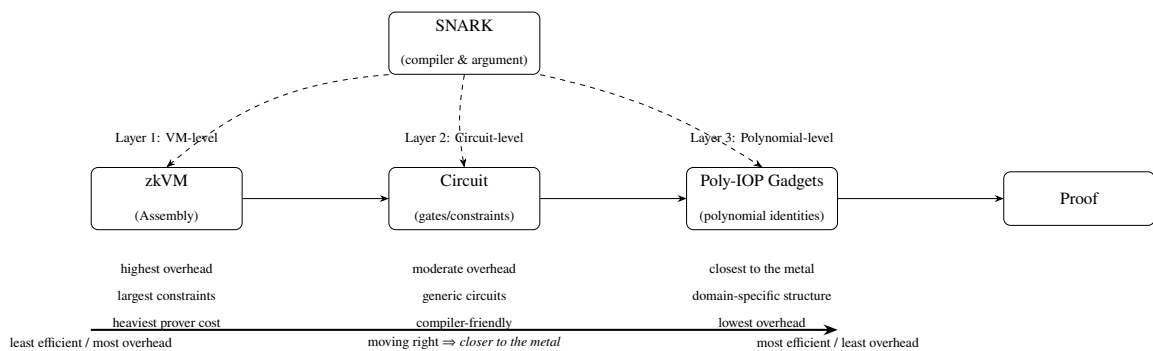


Figure 5.2: Three layers of abstraction for constructing succinct proofs. From left to right, the prover’s statement is expressed at (i) the VM level (zkVM), (ii) the circuit level, or (iii) directly as Poly-IOP gadgets. Shifting right is “closer to the metal” as it exposes more algebraic structure to the proof system, reducing compilation overhead and constraint inflation, and typically yielding more efficient proving and cheaper verification.

Instead of adding these *pre-processing* steps, we implement Zeeperio directly at the Poly-IOP gadget level. This gives us two benefits:

1. **No Compilation:** Circuit compilers and zkVMs add an additional layer of constraints such as range checks, opcode handling, routing selectors and memory abstractions. Since we work directly with Poly-IOP gadgets, we avoid this overhead and complexity entirely.

2. Efficiency: Setting up the polynomial constraints directly optimizes the number of constraints needed to fully verify an election. Instead of verifying thousands of constraints in a circuit or program, Zeeperio reduces to just 18 constraints.

5.1.2 Constraint Enforcement using Vanishing Polynomials

Vanishing polynomials allow us to enforce certain polynomial identities at *selected indices* from the evaluation domain $\Omega = \{\omega^0, \omega^1, \dots, \omega^{n-1}\}$. This is essential for Zeeperio as many of the constraints apply only to a subset of rows.

For example, if we want to enforce a constraint at $\omega^{n-1} \in \Omega$, the vanishing polynomial is defined as:

$$Z(X) = \frac{X^n - 1}{X - \omega^{n-1}}. \quad (5.1)$$

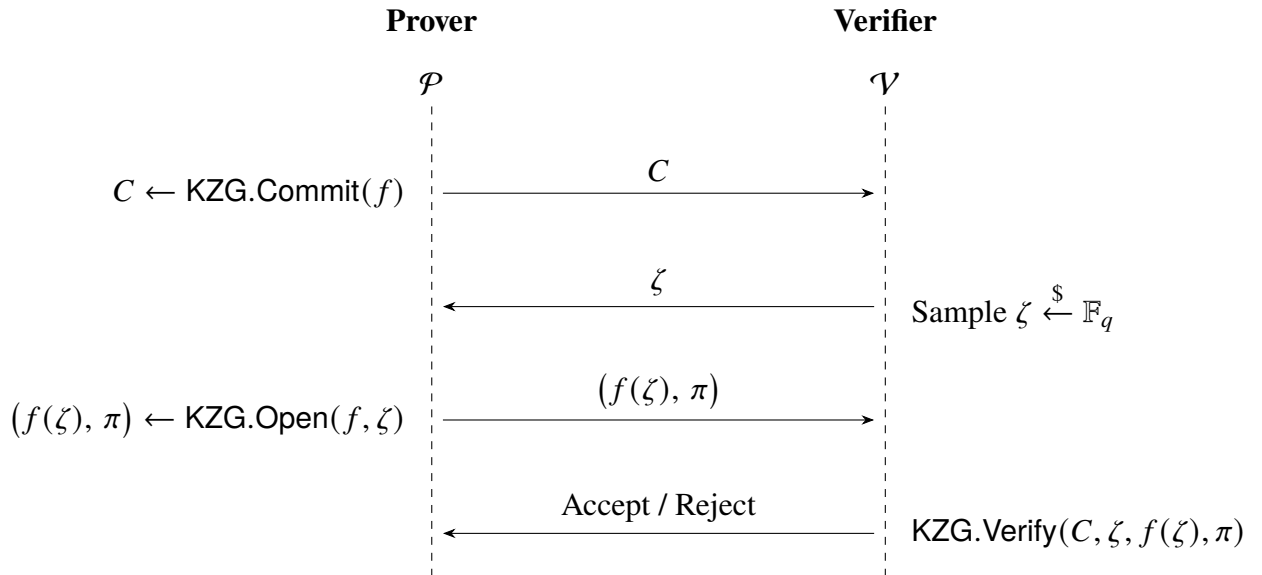
This polynomial zeros out every $X \in \Omega$ except $X = \omega^{n-1}$.

We multiply a constraint polynomial $C(X)$ by $Z(X)$ to enforce the constraint holds only at the specified point. If the constraint holds, $Z(X) \cdot C(X) = 0$ for $X \in \Omega$. Therefore, vanishing polynomials are used as selective *activation masks* for constraints.

5.1.3 On-Chain Verification via Ethereum

In an interactive setting, we design a Poly-IOP that follows a structure shown in Fig. 5.3. The prover first sends all the commitments to the witness polynomials. The verifier samples a random challenge $\zeta \xleftarrow{\$} \mathbb{F}_q$ and the prover responds with evaluation openings at the random ζ . Finally, the verifier either accepts or rejects the proof.

Figure 5.3: Poly-IOP structure



In a non-interactive setting, the sampling of the random ζ is replaced by applying the Fiat–Shamir heuristic [128].

The verification logic is encoded onto an immutable smart contract deployed on Ethereum Mainnet. The smart contract is then verified by every full node in the Ethereum network, with the current node count exceeding 8,000 globally as of Nov 10 2025 [129]. This decentralized setup shifts trust from centralized authorities and hardware to Ethereum’s consensus protocol and the economic guarantees it provides. A successful attack would require collusion among at least one-third of Ethereum’s validators, which is currently secured by just under \$20 billion in collateral as of Nov 10 2025. This makes any potential attack extremely infeasible [130].

5.1.4 Finite Field and Evaluation Domain

All polynomial commitments and operations operate over the prime field \mathbb{F}_q , where q is the order of the BN254 elliptic curve scalar field:

$$q = 36x^4 + 36x^3 + 24x^2 + 6x + 1,$$

for $x = 4965661367192848881$ [131].

The BN254 curve is pairing friendly with 128-bit security, and allows us to perform efficient pairings $e : G_1 \times G_2 \rightarrow G_T$ [131]. BN254 supports SNARK verification on environments with limited resources and the Ethereum Virtual Machine (EVM) provides native opcodes for its curve operations. The field size q defines the **Schwartz-Zippel** bound for our polynomial identity testing. The Schwartz-Zippel lemma states if $P \in \mathbb{F}_q[X]$ is a nonzero polynomial of degree d , then

$$\Pr[P(\zeta) = 0] \leq \frac{d}{q},$$

where $\zeta \in \mathbb{F}_q$ [113]. In Zeeperio’s case, this yields a soundness error $\epsilon \leq \frac{20}{|q|}$ which is negligible given the size \mathbb{F}_q . We discuss the rationale behind choosing degree-20 polynomials in Sec. 5.2.

If n is the evaluation domain size, we denote $\omega \in \mathbb{F}_q$ as a primitive n th root of unity where $\omega^n = 1$ and $\omega^i \neq 1$ for $0 \leq i < n$. For each column $F \in \{\mathbf{B}_{\text{ID}}, \mathbf{A}, \mathbf{M}, \mathbf{C}_{\text{confirm}}\}$, we build a degree $< n$ polynomial $F(X) \in \mathbb{F}_q[X]$ via FFT interpolation over the domain $\Omega = \{\omega^0, \omega^1, \dots, \omega^{n-1}\}$:

$$F(X) = \sum_{i=0}^{n-1} a_i X^i,$$

where the coefficients are calculated by the inverse FFT:

$$(a_0, \dots, a_{n-1}) \xleftarrow{\text{IFFT}} (F[0], \dots, F[n-1]).$$

5.1.5 Zero-Knowledge

So far, we have described Zeeperio as a non-interactive SNARK. However, Zeeperio requires the *zero-knowledge* non-interactive SNARK variant to ensure no information about the votes or confirmation codes is leaked from the published commitments or proofs. We achieve this using the *hiding* variant of the KZG commitment scheme and an off-domain masking technique introduced in *Sonic*, which we call the *Z-Masking* heuristic [132].

1. **Hiding KZG commitments** In the standard KZG scheme, the commitment to a polynomial $F(X) = \sum_{i=0}^{n-1} F_i X^i$ is defined as:

$$g^x \leftarrow \text{KZG.Commit}(x),$$

where $x = F(\tau)$ and $(g, g^\tau, g^{\tau^2}, \dots, g^{\tau^d})$ are elements of the SRS. Because this commitment is not hiding, the verifier can potentially correlate multiple commitments to learn some relationship between the coefficients. To prevent this, Zeeperio uses the *hiding* variant of KZG commitments. For each polynomial $F(X)$, the EA generates a random blinding $r \xleftarrow{\$} \mathbb{F}_q$ using Fiat-Shamir and commits to

$$g^x \cdot h^r \leftarrow \text{KZG.Commit}(x),$$

where h is a fixed generator independent of g . This masks the evaluation of $F(X)$ at all points, making sure the commitments reveal no information about the witness data.

2. **Z-Masking heuristic** Even by using the hiding variant of commitments, the repeated openings of $F(X)$ at random evaluation points in Ω could leak linear information. To account for this, we apply the z-masking heuristic by randomizing each witness polynomial before commitment. This is done by generating a random $\rho \xleftarrow{\$} \mathbb{F}_q$ via

Fiat-Shamir and multiplying it by the domain vanishing polynomial $Z(X)$ as shown below:

$$F'(X) = F(X) + \rho \cdot Z(X). \quad (5.2)$$

Because $Z(X)$ evaluates to zero on the entire evaluation domain Ω , all the constraints remain unchanged, However, this strategy ensures $F'(X)$ takes random, unpredictable values at points outside the domain. This prevents anyone from confirming any polynomial guess even after multiple openings.

Using both these techniques gives us *computational zero-knowledge* for all witness data in Zeeperio. Each committed polynomial is indistinguishable from a random committed polynomial of equal degree, and all openings reveal only the specific constraints checked by the verifier. Throughout the protocol, we assume every witness polynomial to be randomized as Equation 5.2, and commitments to be formed using the hiding KZG variant.

These definitions establish the basic framework on which the Zeeperio protocol is built upon.

5.2 Setup Phase

Like many Poly-IOP-based systems, Zeeperio begins with a trusted setup phase to generate a Structured Reference String (SRS). We use Ethereum’s *Powers of Tau* ceremony, a public multi-party computation that produces a universal and updatable SRS composed of elliptic curve group elements

$$\text{SRS} = \left(g, g^\tau, g^{\tau^2}, \dots, g^{\tau^d} \right),$$

where τ is a secret value discarded after the setup [133]. This SRS supports pairing-based cryptography and is used to commit to polynomials in the KZG scheme.

We use Ethereum’s *degree-20 Powers of Tau* output, which supports polynomials of degree less than $2^{20} = 1,048,576$. This degree was chosen to comfortably accommodate the maximum expected number of voters at any single polling location. With a 20-degree SRS, Zeeperio can support more than one million unique ballot-candidate positions. This ensures there is sufficient capacity even under extreme real-world election conditions. For example, the Kingston and Islands electoral district has 134,415 residents as of 2025 [134]. With a 20-degree SRS, Zeeperio can support this district with up to 8 candidates.

The security of the protocol depends on the infeasibility of recovering τ . In this model, no adversary can forge valid proofs without solving discrete logarithms in the elliptic curve group. Since we directly use Ethereum’s *degree-20 Powers of Tau*, there is no other trust assumption required and all parameters are public and reusable across all elections.

5.3 Audit Column Constraints

The **Audit Column** $A[n]$ is an array of length n that indicates the audit status of ballot \mathcal{B}_j . For any ballot selected for an audit, $A[i] = 1$ for all its C positions. In contrast, if the ballot was cast correctly, $A[i] = 0$ for all of its C positions.

To enforce this, we define a set of constraints that must hold for column A to be formed correctly:

- **Padding:** The padded values $A[i]$ where $i \geq \mathcal{P}$ must be 0.
- **Values:** The column A should contain only 0s or 1s.
- **Blocks:** All C positions on ballot \mathcal{B}_j must be blocks of 0s or 1s.
- **Count:** The EA publishes Sum_A , the total number of ballots that were audited. We need to enforce the number of 1s in column A is equal to $\text{Sum}_A \cdot C$.

We now describe how each of these constraints are enforced using Poly-IOP gadgets.

5.3.1 Padding

We need to show that $A[i] = 0$ for all padded values $i \geq \mathcal{P}$. This can be achieved by zeroing out all non-padded entries at $0 \leq i < \mathcal{P}$. As a result, the audit column polynomial $A(X)$ should evaluate to 0 for every $\omega \in \Omega$.

We define Equation 5.3, a constraint specific vanishing polynomial with roots at $\{\omega^0, \omega^1, \dots, \omega^{\mathcal{P}-1}\}$:

$$Z(X) = \prod_{i=0}^{\mathcal{P}-1} (X - \omega^i). \quad (5.3)$$

If the audit column padding is implemented correctly, $A(X) \cdot Z(X)$ zeros out on the entire evaluation domain Ω . For any $i < \mathcal{P}$, ω^i is a root of $Z(X)$, so $A(\omega^i) \cdot Z(\omega^i) = 0$ regardless of $A(\omega^i)$. For $\mathcal{P} \leq i < n$ (the padded values), $A(\omega^i) \cdot Z(\omega^i) = 0$ iff $A(\omega^i) = 0$. Therefore, the domain vanishing polynomial $P_{\text{vanish1}}(X)$ is defined as:

$$P_{\text{vanish1}}(X) = A(X) \cdot Z(X), \quad (5.4)$$

where $P_{\text{vanish1}}(X) = 0$ on the evaluation domain Ω .

5.3.2 Values

We need to show every entry in column $A[i]$ for $0 \leq i < n$ is either 0 or 1. The goal is to build a vanishing polynomial $P_{\text{vanish2}}(X)$ such that it zeros out on the entire evaluation domain. We can achieve this by setting the roots at $\{0, 1\}$. Therefore, the domain vanishing polynomial $P_{\text{vanish2}}(X)$ is defined as:

$$P_{\text{vanish2}}(X) = A(X) \cdot (A(X) - 1). \quad (5.5)$$

For any $A[i]$ set to 1 at indices $0 \leq i < n$, $A(X)$ is a root of $P_{\text{vanish2}}(X)$. On the other hand, for any $A[i]$ set to 0, $(A(X) - 1)$ is also a root of $P_{\text{vanish2}}(X)$. If $P_{\text{vanish2}}(X)$ zeros out at every $\omega \in \Omega$, the audit column only contains 0s and 1s.

5.3.3 Blocks

For the audit column to be formed correctly, we need to show the audit status for ballot \mathcal{B}_j is consistent among all its C positions. As shown in Fig. 5.1, there are three candidates, so the audit column contains the same value for each block of three. If we add the values of each block, the result should be either 0 (for the cast ballots), or C (for the audited ballots).

We define a block-sum polynomial $S_{\text{blk}_A}(X)$ where every $X = \omega^i \in \Omega$ corresponds to the sum of the C consecutive positions in $A[i]$ for all $0 \leq i < n$ below:

$$S_{\text{blk}_A}(X) = \sum_{k=0}^{C-1} A(\omega^k \cdot X). \quad (5.6)$$

Every evaluation point $X = \omega^{j \cdot C}$ in $S_{\text{blk}_A}(X)$ corresponds to the starting index $k = 0$ of ballot $\mathcal{B}_{\text{ID}_j}$, and evaluates the sum over all its C positions. If the audit column is correctly formed, these positions correspond to 0 or C . Since we only care about the positions where $X = \omega^{j \cdot C}$, we define a selector array $\text{Sel}_{\text{blk}_A}[n]$ that preserves only the specified positions. Because the audited ballots do not require ballot secrecy, the EA knows which ballots are audited and can precompute the array by setting 1 at every $i = j \cdot C$ for $0 \leq i < \mathcal{B}$, and 0 for the rest. This does not depend on voter selections.

Instead of interpolating the selector array using FFT, we can use a sum of the *Lagrange basis* where

$$L_i(\omega^k) = \begin{cases} 1, & \text{if } k = i, \\ 0, & \text{if } k \neq i. \end{cases}$$

We can use this to build the selector polynomial by adding the Lagrange basis at every C th position defined as:

$$\text{Sel}_{\text{blk}_A}(X) = \sum_{j=0}^{\lfloor \frac{n}{C} \rfloor - 1} L_{j \cdot C}(X). \quad (5.7)$$

In this case, interpolation using the Lagrange basis is slightly more efficient ($O(n)$) than using FFT ($O(n \log n)$) [135]. This is because the sum stops at $\lfloor \frac{n}{C} \rfloor - 1$, and the padding bits are set to 0. This also zeros out any tailing irregularities due to the wrap.

We need to prove $\mathbf{S}_{\text{blk}_A}(X)$ and $\text{Sel}_{\text{blk}_A}(X)$ are correctly formed. Since $\text{Sel}_{\text{blk}_A}(X)$ is public and precomputed, we only need to satisfy the following constraint for $\mathbf{S}_{\text{blk}_A}(X)$:

$$\mathbf{S}_{\text{blk}_A}(\omega^{j \cdot C}) = \sum_{i=0}^{C-1} \mathbf{A}(\omega^{j \cdot C + i}),$$

for all blocks $j = \{0, \dots, \lfloor \frac{n}{C} \rfloor - 1\}$.

We can convert this constraint into the domain vanishing polynomial defined as:

$$\mathbf{P}_{\text{vanish3}}(X) = (\mathbf{S}_{\text{blk}_A}(X) - \sum_{i=0}^{n-1} \mathbf{A}(X \cdot \omega^i)) \cdot \text{Sel}_{\text{blk}_A}(X), \quad (5.8)$$

over the evaluation domain Ω .

Finally, we can use the same strategy used in Sec. 5.3.2 to prove the evaluation of $\mathbf{S}_{\text{blk}_A}(X) \cdot \text{Sel}_{\text{blk}_A}(X)$ corresponds to 0 or C . We form the domain vanishing polynomial defined as:

$$\mathbf{P}_{\text{vanish4}}(X) = (\mathbf{S}_{\text{blk}_A}(X) \cdot \text{Sel}_{\text{blk}_A}(X)) \cdot (\mathbf{S}_{\text{blk}_A}(X) \cdot \text{Sel}_{\text{blk}_A}(X) - C), \quad (5.9)$$

over the evaluation domain Ω .

5.3.4 Count

Let Sum_A be the publicly declared number of audited ballots. Given our previous constraint, the total number of 1s in the audit column A should be $\text{Sum}_A \cdot C$. We can prove they are equal using accumulator polynomials. We define a backwards accumulator $\text{Acc}_A(X)$ that encodes the running total of the audit markings. If the audit column is correctly formed, $\text{Acc}_A(\omega^0) = \text{Sum}_A \cdot C$.

We start by setting the last position of the accumulator polynomial to equal $A(X)$ evaluated at the last position such that $\text{Acc}_A(\omega^{n-1}) = A(\omega^{n-1})$. For each preceding index i , we set $\text{Acc}_A(X) = A(X) + \text{Acc}_A(\omega \cdot X)$, where $X = \omega^i$ and the preceding index $\omega \cdot X = \omega^{i+1}$ in the evaluation domain Ω .

To prove these relationships, we need to enforce the following three constraints over the domain Ω .

1. $\text{Acc}_A(X) = A(X) + \text{Acc}_A(\omega X)$ for all $X = \omega^i$ except $X = \omega^{n-1}$

To enforce this constraint, we define a constraint specific vanishing polynomial $Z_{1_A}(X)$ that zeros out the last position at $X = \omega^{n-1}$ as:

$$Z_{1_A}(X) = X - \omega^{n-1}. \quad (5.10)$$

Since this allows us to preserve all values at $\omega \in \Omega$ except ω^{n-1} , we can check if the relationship holds. We define the domain vanishing polynomial as:

$$P_{\text{vanish5}}(X) = (\text{Acc}_A(X) - A(X) + \text{Acc}_A(\omega \cdot X)) \cdot Z_1(X). \quad (5.11)$$

2. $\text{Acc}_A(\omega^{n-1}) = A(\omega^{n-1})$ for $X = \omega^{n-1}$

We need to prove the last position of $\text{Acc}_A(X) = A(X)$. To enforce this, we define a constraint specific vanishing polynomial $Z_{2_A}(X)$ that zeros out every $\omega \in \Omega$ except

ω^{n-1} as:

$$Z_{2_A}(X) = \frac{X^n - 1}{X - \omega^{n-1}}. \quad (5.12)$$

This vanishing polynomial lets us preserve only the last position where $i = n - 1$. We define the domain vanishing polynomial as:

$$P_{\text{vanish6}}(X) = (\text{Acc}_A(X) - A(X)) \cdot Z_2(X). \quad (5.13)$$

3. $\text{Acc}_A(\omega^0) = \text{Sum}_A \cdot C$ for $X = \omega^0$

If the accumulator is built correctly, the evaluation at $X = \omega^0$ should be the number of 1s in the audit column. We only want to preserve the first position, so we define a constraint specific vanishing polynomial $Z_{3_A}(X)$ that zeros out every $\omega \in \Omega$ except ω^0 as:

$$Z_{3_A}(X) = \frac{X^n - 1}{X - \omega^0}. \quad (5.14)$$

We define the domain vanishing polynomial as:

$$P_{\text{vanish7}}(X) = (\text{Acc}_A(X) - \text{Sum}_A \cdot C) \cdot \frac{X^n - 1}{X - \omega^0}. \quad (5.15)$$

These polynomials *vanish* on the entire evaluation domain iff the accumulator was correctly formed.

5.4 Constraint: Mark Column

The **Mark Column** $M[n]$ is an array of length n that indicates which position on ballot \mathcal{B}_j was voted for. For any ballot that was cast correctly, $M[i] = 1$ for exactly one of its position and all others 0 to prevent overvotes. If a ballot was audited instead, then $M[i] = 0$ for all of its positions.

We apply the same constraints to the mark column to prove it was correctly formed.

- **Padding:** $M[i] = 0$ for all the padded positions $\mathcal{P} \leq i < n$. We use the same constraint specific vanishing technique as Sec. 5.3.1, but apply it to $M(X)$. The resulting domain vanishing polynomial is defined as:

$$P_{\text{vanish8}}(X) = M(X) \cdot Z(X), \quad (5.16)$$

where $Z(X) = \prod_{i=0}^{\mathcal{P}-1} (X - \omega^i)$.

- **Values:** The mark column M should only contain 0s and 1s. We use the same technique as Sec. 5.3.2, but apply it to $M(X)$. The resulting domain vanishing polynomial is defined as:

$$P_{\text{vanish9}}(X) = M(X) \cdot (M(X) - 1). \quad (5.17)$$

- **Blocks:** We need to enforce there is at most one marked position per ballot. Similar to Sec. 5.3.3, we build a block-sum polynomial $S_{\text{blk}_M}(X)$ that sums over the C consecutive positions. This constraint is easier to enforce compared to the audit column counterpart as we need to only verify $S_{\text{blk}_M}(X)$ corresponds to $\{0, 1\}$ rather than $\{0, 1, \dots, C\}$. This is why we do not need the selector polynomial for the mark column.

We define the block-sum polynomial as:

$$S_{\text{blk}_M}(X) = \sum_{k=0}^{C-1} M(\omega^k \cdot X). \quad (5.18)$$

If there is only one mark per ballot, the evaluations of $S_{\text{blk}_M}(X)$ on the domain Ω correspond to 0 or 1. We need to prove $S_{\text{blk}_M}(X)$ was constructed correctly, so we

define a vanishing polynomial that satisfies this constraint as:

$$P_{\text{vanish10}}(X) = (\mathbf{S}_{\text{blk}}(X) - \sum_{i=0}^{n-1} M(X \cdot \omega^i)). \quad (5.19)$$

Now we need to show $\mathbf{S}_{\text{blk}_M}(X)$ corresponds to $\{0, 1\}$. Enforcing this constraint is identical to enforcing the *values* constraint. We multiply the roots to form the domain vanishing polynomial defined as:

$$P_{\text{vanish11}}(X) = \mathbf{S}_{\text{blk}_M}(X) \cdot (\mathbf{S}_{\text{blk}_M}(X) - 1). \quad (5.20)$$

This zeros out the polynomial on the evaluation domain iff the mark column contains at most one mark per every C th block.

- **Count:** Let Sum_M be the public number of cast ballots. We want to prove the total number of marked positions in the column M is equal to Sum_M , since each cast ballot counts as one mark.

We use the same accumulation technique from Sec. 5.3.4 to sum over all positions in $M(X)$. We define the accumulator polynomial $\text{Acc}_M(X)$ such that $\text{Acc}_M(\omega^{n-1}) = M(\omega^{n-1})$, and $\text{Acc}_M(X) = A(X) + \text{Acc}_M(\omega \cdot X)$ for each preceding index i .

We need to show the accumulator polynomial $\text{Acc}_M(X)$ is formed correctly and $\text{Acc}_M(X)$ evaluated at $X = \omega^0$ corresponds to Sum_M . There are three constraints we must satisfy:

1. $\text{Acc}_M(X) = M(X)$ for $X = \omega^{n-1}$
2. $\text{Acc}_M(X) = \text{Sum}_M$ for $X = \omega^0$
3. $\text{Acc}_M(X) = M(X) + \text{Acc}_M(\omega \cdot X)$ for all X except $X = \omega^{n-1}$

We can use the same constraint specific vanishing technique to control which positions

to enforce and form the following domain vanishing polynomials defined by Equations 5.21 to 5.23.

$$P_{\text{vanish12}}(X) = (\text{Acc}_M(X) - M(X) + \text{Acc}_M(\omega \cdot X)) \cdot Z_{1M}(X) \quad (5.21)$$

$$P_{\text{vanish13}}(X) = (\text{Acc}_M(X) - M(X)) \cdot Z_{2M}(X) \quad (5.22)$$

$$P_{\text{vanish14}}(X) = (\text{Acc}_M(X) - \text{Sum}_M) \cdot Z_{3M}(X) \quad (5.23)$$

where

$$Z_{1M}(X) = X - \omega^{n-1}, Z_{2M}(X) = \frac{X^n - 1}{X - \omega^{n-1}}, \text{ and } Z_{3M}(X) = \frac{X^n - 1}{X - \omega^0}.$$

5.5 Constraint: Audit and Mark Column Exclusions

So far, we have enforced constraints that make sure the audit column A and mark column M are formed correctly. Now, we need to enforce no ballot is marked as both audited and cast. Essentially, we need to prove both A and M are not set to 1 at the same index $0 \leq i < n$.

This can be easily achieved by calculating the **Hadamard product** (element-wise multiplication) of the two columns. We define the domain vanishing polynomial as:

$$P_{\text{vanish15}}(X) = A(X) \cdot M(X), \quad (5.24)$$

such that it zeros out on the evaluation domain Ω iff at least one of the columns is set to 0. If a ballot is audited, the corresponding audit column positions are set to 1, and the corresponding mark column positions are set to 0. On the other hand, if a ballot is cast, only one of the corresponding positions in the mark column is set to 1, and all the corresponding positions in the audit column are set to 0.

5.6 Constraint: Tally Check

Finally, we need to verify the final election tally (the number of votes for each candidate) is correctly calculated from the mark column M . As shown in Fig. 5.1, the candidates on each ballot are in a fixed canonical order where each candidate corresponds to the same position on each ballot. Therefore, we can derive the total count for each candidate by grouping the elements of M by candidate position across all ballots.

For candidate $c \in \{0, \dots, C - 1\}$ let us consider the indices $\{c, c + C, c + 2 \cdot C, \dots\}$ where these indices represent the c th position on every ballot. We will refer to these indices as a *stream*, and use the term *stride* for skipping ahead C indices.

We want to prove the sum of M over the stream is equal to the tally for candidate c . We can calculate the sum for each candidate all in one step by using a strided accumulation. This is similar to accumulator polynomial used to prove $\text{Sum}_M = \text{Acc}_M(\omega^0)$ in Sec. 5.4, except we shift by ω^C on the domain instead of ω .

In a normal sum:

$$(\text{Acc}_M(X) - M(X) + \text{Acc}_M(\omega \cdot X)) \cdot Z(X) = 0.$$

In a strided sum:

$$(\text{Acc}_M(X) - M(X) + \text{Acc}_M(\omega^C \cdot X)) \cdot Z(X) = 0.$$

In a standard accumulator, the sum can include the padded bits because the padding value is 0, which does not affect the tally.

In a strided accumulator, we can do the same thing but we zero out the last C elements. If n is not divisible by C , there will be some fully padded blocks for which all its C positions are

set to 0, and some remaining padded bits that do not form a full block of C positions. We call these residual bits the *padding tail*. This means some streams start from the padding tail and others start from the last padding block. However, this is not an issue as the former streams will just be longer by one zeroed value.

We define the following three constraints to satisfy the tally for each candidate:

1. The last C positions of the accumulator must match the corresponding values from the mark column M :

$$\text{Acc}_M(X) - M(X) = 0 \text{ for } \omega^{n-1-C} \leq X \leq \omega^{n-1}.$$

2. The accumulator must be in the form:

$$\text{Acc}_M(X) - M(X) + \text{Acc}_M(\omega^C \cdot X) = 0 \text{ for all } X \text{ except } \omega^{n-1-C} \leq X \leq \omega^{n-1}.$$

3. The first position of each candidate C in the accumulator must match the publicly announced tally for candidate $C \in \{0, 1, \dots, C - 1\}$

$$\text{Acc}_M(X) - \text{Sum}_{C_i} = 0 \text{ for each } \omega^0 \leq X \leq \omega^{C-1}.$$

It is important to note this constraint is applied to every candidate separately. Therefore, there will be C vanishing polynomials associated with this constraint.

We can turn these constraints into domain vanishing polynomials defined as Equations 5.25 to 5.27.

$$P_{\text{vanish16}}(X) = (\text{Acc}_M(X) - M(X)) \cdot Z_1(X) \tag{5.25}$$

where $Z_1(X) = \prod_{i=0}^{C-1} (X - \omega^{n-1-i})$.

$$P_{\text{vanish17}}(X) = (\text{Acc}_M(X) - M(X) + \text{Acc}_M(\omega^C X)) \cdot Z_2(X) \quad (5.26)$$

where $Z_2(X) = \left(\prod_{i=0}^{C-1} \frac{X^{n-1}}{X - \omega^{n-1-i}} \right)$.

$$\{P_{\text{vanish18},c}(X)\}_{c=0}^{C-1} = (\text{Acc}_M(X) - \text{Sum}_c) \cdot Z_3(X) \text{ for each } \omega^0 \leq X \leq \omega^{C-1} \quad (5.27)$$

where $Z_3(X) = \frac{X^{n-1}}{X - \omega^c}$.

5.7 Election Verification

Now that we have fully defined the election constraints, we will show how the election verification is performed.

First, the EA commits to all polynomials involved in the vanishing constraints, including the public and private witness polynomials:

$$\text{Commitments} = \left[\begin{array}{ccc} \text{KZG.Commit}(A(X)), & \text{KZG.Commit}(M(X)), & \text{KZG.Commit}(S_{\text{blk}_A}(X)), \\ \text{KZG.Commit}(S_{\text{blk}_M}(X)), & \text{KZG.Commit}(\text{Acc}_A(X)), & \text{KZG.Commit}(\text{Acc}_M(X)). \end{array} \right]$$

Then, the EA batches the $k = 17 + C$ vanishing polynomials into a single polynomial using **random linear combination**. Instead of verifying each constraint separately, the EA generates a random $\alpha \in \mathbb{F}_q$ via Fiat-Shamir and builds the new domain vanishing polynomial defined as:

$$P_{\text{vanish}}(X) = \sum_{i=1}^k (\alpha^{i-1} \cdot P_{\text{vanish}_i}(X)). \quad (5.28)$$

This only works because KZG commitments are homomorphic. This allows for us to

perform operations on committed polynomials without needing to reveal the underlying data.

Next, the EA constructs the quotient polynomial $Q(X)$ defined as:

$$Q(X) = \frac{P_{\text{vanish}}(X)}{X^n - 1}. \quad (5.29)$$

$Q(X)$ exists iff all constraints encoded in $P_{\text{vanish}}(X)$ evaluate to zero over the domain Ω . Otherwise, we end up with a rational function which we cannot commit to using the KZG scheme. Therefore, the existence of $Q(X)$ proves our election is enforced correctly.

The EA then commits to $Q(X)$ to produce the proof π :

$$\pi \leftarrow \text{KZG.Commit}(Q(X)).$$

The EA generates a challenge $\zeta \xleftarrow{\$} \mathbb{F}_q$ via Fiat-Shamir, and sends all relevant commitment openings at the evaluation point ζ to the verifier:

$$\text{Openings} = \left[\begin{array}{l} \text{KZG.Open}(Q(\zeta)), \text{KZG.Open}(A(\zeta)), \text{KZG.Open}(M(\zeta)), \\ \text{KZG.Open}(S_{\text{blk}_A}(\zeta)), \text{KZG.Open}(S_{\text{blk}_M}(\zeta)), \\ \text{KZG.Open}(\text{Acc}_A(\zeta)), \text{KZG.Open}(\text{Acc}_M(\zeta)). \end{array} \right]$$

Finally, the verifier builds $P_{\text{vanish}}(X)$ from all the commitment openings at ζ and verifies $P_{\text{vanish}}(\zeta) - Q(\zeta) \cdot (\zeta^n - 1) \stackrel{?}{=} 0$ using the pairing-based KZG.Verify . If this check holds, then by the Schwartz-Zippel lemma we can conclude this relationship holds for any $X \in \mathbb{F}_q$ with overwhelming probability. Thus, we are assured that all Zeeperio election constraints are satisfied without learning any private ballot data.

5.8 Constraint: Voter Verification

So far, we have verified the election correctness through the constraint system. Now, we focus on the voter verification. This ensures that any data revealed to voters is consistent with the committed election data. This is a separate, independent, zero-knowledge proof from the main election proof used specifically in the event a voter raises a dispute. The underlying setup phase remains the same, and we continue to operate over the same election parameters and committed polynomials.

5.8.1 Print Audit

In a print audit, an auditor selects a ballot to spoil and provides the ballot ID to the EA. The EA must prove the ballot corresponds to the committed values in $A[i]$ and $M[i]$. Because ballot secrecy is not a concern for audited ballots, the EA can simply use KZG openings to reveal the commitments of the corresponding columns. If given a ballot \mathcal{B}_j , the EA identifies the corresponding block of indices $i = [j \cdot C, j \cdot C + 1, \dots, j \cdot C + (C - 1)]$. It then opens the commitments to $A[i]$ at those index values using KZG to show all of them are marked as 1s (confirms ballot was marked as audited). The EA also opens commitments to $M[i]$ at those positions to show they are all 0 (confirms none of the positions were used to vote). Since we do not have to preserve zero-knowledge with audited ballots, this is enough to satisfy the print audit constraint.

5.8.2 Voter Checks

After casting a ballot, the voter receives a receipt with a unique confirmation code $C_{\text{confirm}}[i]$. If the voter raises a dispute, they provide their ballot ID \mathcal{B}_j and the confirmation code C . We consider two scenarios:

1. **Ballot inclusion ("Was my ballot included?")** The EA proves the confirmation code

C is associated with ballot \mathcal{B}_j and appears in the committed election data.

2. **Receipt correctness ("Is my confirmation code correct?")** The voter can raise a dispute if they do not see their confirmation code in the published $C_{\text{confirm}}(X)$ data. The EA proves the disputed confirmation code C is not associated with any of the positions on the ballot \mathcal{B}_j .

Ballot Inclusion

A simple receipt check follows the same logic as Sec. 5.8.1, but uses the mark column instead of the audit column. However, this reveals which candidate was voted for and therefore is not receipt-free. Instead, we follow the approach of Eperio [1] and use a permutation argument to prove the voter's confirmation code C is associated with a marked position on their ballot \mathcal{B}_j , while keeping the candidate chosen hidden.

Tuples and Shuffles

For each index $i \in \{0, \dots, n-1\}$, the EA forms a tuple

$$(b_i, c_i, m_i) = (B_{\text{ID}}[i], C_{\text{confirm}}[i], M[i])$$

using the ballot ID, confirmation code, and mark columns. Let σ be a Fiat-Shamir derived shuffling algorithm applied to each tuple (b_i, c_i, m_i) . We define the permuted tuples as

$$(b'_i, c'_i, m'_i) = (b_{\sigma(i)}, c_{\sigma(i)}, m_{\sigma(i)})$$

and denote the corresponding original and shuffled interpolated polynomials as

$$(B_{\text{ID}}(X), C_{\text{confirm}}(X), M(X)), \text{ and } (B'_{\text{ID}}(X), C'_{\text{confirm}}(X), M'(X)) \in \mathbb{F}_q[X].$$

The EA then commits to all six polynomials shown below.

$$C = \begin{bmatrix} \text{KZG.Commit}(B_{\text{ID}}), & \text{KZG.Commit}(C_{\text{confirm}}), & \text{KZG.Commit}(M(X)), \\ \text{KZG.Commit}(B'_{\text{ID}}), & \text{KZG.Commit}(C'), & \text{KZG.Commit}(M'). \end{bmatrix}$$

Random linear combination

We use random linear combination to compare the two multisets of tuples without revealing the underlying data. The EA generates $\alpha \xleftarrow{\$} \mathbb{F}_q$ via Fiat-Shamir and defines the multisets below:

$$v_i = b_i + \alpha \cdot c_i + \alpha^2 \cdot m_i \quad (5.30)$$

$$v'_i = b'_i + \alpha \cdot c'_i + \alpha^2 \cdot m'_i \quad (5.31)$$

We define two polynomials $T(X)$ and $T'(X)$ such that $T(\omega^i) = v_i$ and $T'(\omega^i) = v'_i$ for the entire evaluation domain Ω as shown below:

$$T(X) = \alpha \cdot B_{\text{ID}}(X) + \alpha^2 \cdot C_{\text{confirm}}(X) + M(X), \quad (5.32)$$

$$T'(X) = \alpha \cdot B'_{\text{ID}}(X) + \alpha^2 \cdot C'_{\text{confirm}}(X) + M'(X). \quad (5.33)$$

Now we can compare the encoded polynomials by using the grand-product argument over a random point r . We introduce this random point because different sets can produce the same product without containing the same elements. Therefore, we evaluate the products over a random point and if they are equal, we can conclude the tuples $(B_{\text{ID}}, C_{\text{confirm}}, M)$ and $(B'_{\text{ID}}, C'_{\text{confirm}}, M')$ are valid permutations of each other.

The EA generates a random $r \xleftarrow{\$} \mathbb{F}_q$ via Fiat-Shamir and defines $T_1(X)$ and $T_2(X)$ as shown below:

$$T_1(X) = r - T(X), \quad (5.34)$$

$$\mathbb{T}_2(X) = r - \mathbb{T}'(X), \quad (5.35)$$

for the entire evaluation domain $X \in \Omega$.

Grand Product Argument

We now enforce the grand product check using backward accumulators defined as:

$$\text{Acc}_{T_1}(X) = \begin{cases} \text{Acc}_{T_1}(\omega^{n-1}) = \mathbb{T}_1(\omega^{n-1}) \\ \text{Acc}_{T_1}(\omega^i) = \mathbb{T}_1(\omega^i) \cdot \text{Acc}_{T_1}(\omega^{i+1}), \end{cases} \quad (5.36)$$

$$\text{Acc}_{T_2}(X) = \begin{cases} \text{Acc}_{T_2}(\omega^{n-1}) = \mathbb{T}_2(\omega^{n-1}) \\ \text{Acc}_{T_2}(\omega^i) = \mathbb{T}_2(\omega^i) \cdot \text{Acc}_{T_2}(\omega^{i+1}). \end{cases} \quad (5.37)$$

This enforces the equality of products because $\text{Acc}_{T_1}(\omega^0) = \prod_{i=0}^{n-1} \mathbb{T}_1(\omega^i)$ and $\text{Acc}_{T_2}(\omega^0) = \prod_{i=0}^{n-1} \mathbb{T}_2(\omega^i)$. We can then compare if $\text{Acc}_{T_1}(\omega^0) \stackrel{?}{=} \text{Acc}_{T_2}(\omega^0)$. If this check passes, we can conclude the two multisets are equal, and therefore the voter's confirmation code C is associated with a marked position on their ballot \mathcal{B}_j .

To show this equality, we enforce the following five constraints:

1. $\text{Acc}_{T_1}(X) = \mathbb{T}_1(X)$ for $X = \omega^{n-1}$

This constraint verifies the last value of $\text{Acc}_{T_1}(X)$ matches the last value of $\mathbb{T}_1(X)$.

We define the constraint specific vanishing polynomial $Z_1(X)$ that zeros all $\omega \in \Omega$, except ω^{n-1} defined as:

$$Z_1(X) = \frac{X^n - 1}{X - \omega^{n-1}}. \quad (5.38)$$

This ensures the check is only performed on the last position. The resulting domain vanishing polynomial is defined as:

$$P_{\text{vanish1}}(X) = (\text{Acc}_{T_1}(X) - \mathbb{T}_1(X)) \cdot Z_1(X). \quad (5.39)$$

2. $\text{Acc}_{T_2}(X) = T_2(X)$ for $X = \omega^{n-1}$

Similarly, this constraint verifies the last value of $\text{Acc}_{T_2}(X)$ matches the last value of $T_2(X)$. The resulting domain vanishing polynomial is defined as:

$$P_{\text{vanish2}}(X) = (\text{Acc}_{T_2}(X) - T_2(X)) \cdot Z_1(X). \quad (5.40)$$

3. $\text{Acc}_{T_1}(X) = T_1(X) \cdot \text{Acc}_{T_1}(\omega \cdot X)$ for all X except $X = \omega^{n-1}$

This constraint verifies the rest of the accumulator $\text{Acc}_{T_1}(X)$ is formed correctly. We define the constraint specific vanishing polynomial $Z_2(X)$ that only zeros out the last element at $X = \omega^{n-1}$ defined as:

$$Z_2(X) = X - \omega^{n-1}. \quad (5.41)$$

This ensures the check is performed on all $\omega \in \Omega$, except the last element ω^{n-1} . The resulting domain vanishing polynomial is defined as:

$$P_{\text{vanish3}}(X) = (\text{Acc}_{T_1}(X) - T_1(X) \cdot \text{Acc}_{T_1}(\omega \cdot X)) \cdot Z_2(X). \quad (5.42)$$

4. $\text{Acc}_{T_2}(X) = T_2(X) \cdot \text{Acc}_{T_2}(\omega \cdot X)$ for all X except $X = \omega^{n-1}$

Similarly, this constraint verifies the rest of the accumulator $\text{Acc}_{T_2}(X)$ is formed correctly. The resulting domain vanishing polynomial is defined as:

$$P_{\text{vanish4}}(X) = (\text{Acc}_{T_2}(X) - T_2(X) \cdot \text{Acc}_{T_2}(\omega \cdot X)) \cdot Z_2(X). \quad (5.43)$$

5. $\text{Acc}_{T_1}(X) = \text{Acc}_{T_2}(X)$ for $X = \omega^0$

Finally, the last constraint verifies the two accumulators are equal at the first position.

We define the constraint specific vanishing polynomial $Z_3(X)$ that zeros all $\omega \in \Omega$,

except ω^0 defined as:

$$Z_3(X) = \frac{X^n - 1}{X - \omega^0}. \quad (5.44)$$

This ensures the check is only performed on the first element. The resulting domain vanishing polynomial is defined as:

$$P_{\text{vanish5}}(X) = (\text{Acc}_{T_1}(X) - \text{Acc}_{T_2}(X)) \cdot Z_3(X). \quad (5.45)$$

These constraints collectively ensure the accumulators are correctly formed and $\prod_{i=0}^{n-1} T_1(\omega^i) = \prod_{i=0}^{n-1} T_2(\omega^i)$.

We still need to verify $T_1(X)$ and $T_2(X)$ are formed correctly with the random value r to complete the final check. Therefore, we must satisfy the following two constraints.

1. $T_1(X) = r - T(X)$ for all $\omega^0 \leq X \leq \omega^{n-1}$

We can turn this into the domain vanishing polynomial defined as:

$$P_{\text{vanish6}}(X) = T_1(X) - (r - T(X)). \quad (5.46)$$

2. $T_2(X) = r - T'(X)$ for all $\omega^0 \leq X \leq \omega^{n-1}$

We can turn this into the domain vanishing polynomial defined as:

$$P_{\text{vanish7}}(X) = T_2(X) - (r - T'(X)). \quad (5.47)$$

Verification

Verifying all constraints efficiently requires the EA to batch all $k = 7$ vanishing polynomials into a single polynomial. The EA generates a random challenge $\beta \xleftarrow{\$} \mathbb{F}_q$ via Fiat-Shamir and defines the batched vanishing polynomial as:

$$P_{\text{vanish}}(X) = \sum_{i=1}^k \beta^{i-1} \cdot P_{\text{vanish}_i}(X). \quad (5.48)$$

$P_{\text{vanish}}(X)$ vanishes on the evaluation domain Ω iff all constraints (5.39) to (5.47) hold.

The EA then forms the quotient polynomial defined as:

$$Q(X) = \frac{P_{\text{vanish}}(X)}{X^n - 1}. \quad (5.49)$$

Finally, the EA generates an evaluation challenge $\zeta \xleftarrow{\$} \mathbb{F}_q$ via Fiat-Shamir, opens all required committed polynomials at ζ , and the verifier checks Equation 5.50.

$$P_{\text{vanish}}(\zeta) - Q(\zeta) Z_H(\zeta) \stackrel{?}{=} 0 \quad (5.50)$$

If this holds, we can assume the tuples are valid permutations of each other by the Schwartz-Zippel lemma. It is important to note this argument leaks nothing about the mapping of any of the tuples, or the shuffling algorithm σ .

Ballot Check

Once the permutation check has been established, it can be reused for multiple receipt checks. The EA looks for the C tuples where the shuffled ballot ID B'_{ID} matches the ballot ID \mathcal{B}_j provided by the voter. The EA reveals each of the C tuples using KZG openings on the voter's confirmation code. Exactly one of these tuples has a mark value of 1 which reveals the corresponding $(B_{\text{ID}}, C_{\text{confirm}}, 1)$. This proves to the voter their ballot was included in the tally. However, because of the shuffle, the voter does not learn the exact position where the mark was placed, thereby preserving receipt-freeness.

5.8.3 Receipt Correctness

If a voter believes their posted confirmation code does not match the code provided on their receipt, they may raise a dispute. In this case, the EA must prove the disputed code C does not appear in the block of confirmation codes associated with the voter's ballot \mathcal{B}_j . This is all done without revealing any of the codes associated with ballot \mathcal{B}_j .

Since the ballots are in canonical order as shown in Fig. 5.1, the EA can create a selector vector Sel_j such that it *activates* the C positions associated with \mathcal{B}_j . Let $\text{Sel}[n]$ be the selector vector that *activates* indices $[\mathcal{B}_j \cdot C, \mathcal{B}_j \cdot C + 1, \dots, \mathcal{B}_j \cdot C + (C - 1)]$ in the evaluation domain Ω by setting the output as 1, and 0 elsewhere. We do not need to prove $\text{Sel}[n]$ as it is fully determined by the canonical ordering of the election columns.

Equality test

The EA defines a helper vector $D[n] \in \mathbb{F}_q^n$ as:

$$D_i = 1 - (C_{\text{confirm}_i} - C)^{q-1}, \quad (5.51)$$

for $0 \leq i < n$, where q is the (prime) size of the field \mathbb{F}_q .

By Fermat's little theorem, if $a \in \mathbb{F}_q^\times$, then $a^{q-1} = 1$, while $0^{q-1} = 0$ [136]. Hence:

$$C_{\text{confirm}_i} = C \rightarrow (C_{\text{confirm}_i} - C)^{q-1} = 0 \rightarrow D_i = 1 \quad (5.52)$$

$$C_{\text{confirm}_i} \neq C \rightarrow (C_{\text{confirm}_i} - C)^{q-1} = 1 \rightarrow D_i = 0 \quad (5.53)$$

These evaluations are performed locally. It is important to note this operation reveals nothing about any of the confirmation codes associated with ballot \mathcal{B}_j .

The EA commits to the helper polynomial $D(X)$ to prove it is correctly formed. It generates a random challenge $\zeta \in \mathbb{F}_q$ via Fiat-Shamir and reveals $C(\zeta)$, and $D(\zeta)$ for the verifier to

check the relationship below:

$$D(\zeta) \stackrel{?}{=} 1 - (C_{\text{confirm}_i}(\zeta) - C)^{q-1}.$$

If this holds, then by the Schwartz-Zippel lemma we can assume the helper array D was formed correctly.

Disputed code verification

The EA needs to show the disputed code C does not correspond to the voter's ballot \mathcal{B}_j . As shown in (5.52), $D(X)$ is set to 1 at every index where the committed code matches C .

We isolate these matches by summing the multiplication of $D(X)$ and $\text{Sel}(X)$. We define the sum z below:

$$z = \sum_{i=0}^{n-1} \text{Sel}(X) \cdot D(X). \quad (5.54)$$

Because $\text{Sel}(X)$ corresponds to 1 exactly on the indices of ballot \mathcal{B}_j , z sums the number of positions on that ballot where $\text{confirm}_i = C$. If $z = 0$, the disputed code c does not appear on ballot \mathcal{B}_j .

This shows that no committed confirmation code on the voter's ballot matches the disputed code C , all without revealing which codes actually appear on ballot \mathcal{B}_j .

Chapter 6

6 Zeeperio Proofs of Security, Implementation, Evaluation, and Future Work

In this chapter, we first provide security proofs for Zeeperio, showing that the encoded zk-SNARK constraints are equivalent to the original Eperio election conditions. Next, we go over Zeeperio’s implementation, divided into the off-chain *prover* and the on-chain *verifier*. We show the results from a sample 100,000 ballot, 5 candidate election. We discuss the feasibility of on-chain verification, and analyze any trade-offs. Finally we discuss future work and possible extensions to Zeeperio.

6.1 Proofs of Security

In this section, we formally prove that Zeeperio’s SNARK is *faithful* to the original Eperio protocol. Prior work has already established Eperio’s E2E verifiability under its stated assumptions [1, 21]. Poly-IOP-based SNARKs provide (knowledge) completeness, soundness, and (when required) zero knowledge for relations expressed as low-degree polynomial constraints that are checked via vanishing tests at random challenges [124]. Zeeperio uses this framework by encoding Eperio’s table semantics as polynomial identities over an evaluation domain. The remaining link is *constraint faithfulness*. We need to ensure that Zeeperio’s constraints match Eperio’s logic for tally correctness, voter inclusion, and dispute resolution.

To do this, we show two different families of proofs:

- **Soundness:** If Zeeperio’s proofs are verified, then the underlying election data satisfies Eperio’s conditions of tally correctness, voter inclusion, and dispute resolution.
- **Completeness:** If the election is honest and satisfies those Eperio conditions, then

one can construct a corresponding Zeeperio proof that will verify as well.

Together, these proofs show that Zeeperio’s constraints enforce *exactly* the intended Eperio integrity properties, which we call *constraint faithfulness*.

We begin by recalling the election table conditions taken from Eperio, and the corresponding Zeeperio proofs. An election with ballots \mathcal{B}_j (for $j \in \{0, \dots, \mathcal{B} - 1\}$) and C candidates per ballot is represented by columns. The witness columns include the *Mark* column $M[i] \in \{0, 1\}$, the *Audit* column $A[i] \in \{0, 1\}$, and the *Confirmation code* column $C_{\text{confirm}}[i] \in \mathbb{F}_q$. Zeeperio commits to these columns (with padding rows set to zero) and proves that they satisfy the same validity semantics as Eperio’s table checks.

The following three properties fully capture the validity conditions of Eperio that Zeeperio must enforce:

- **Tally correctness:**

Let $\text{Tally}[j]$ denote the publicly announced tally for candidate $j \in \{0, \dots, C - 1\}$. The tally must equal the number of marked votes for that candidate among *cast* (non-audited) ballots. In Zeeperio, this is enforced by two separate constraints. The first constraint (ballot-level) enforces no audited ballots contribute to any mark ($A \cdot M = 0$, audited blocks have all $M = 0$). The second constraint uses a strided accumulation that enforces

$$\text{Tally}[j] = \sum_{i=0}^{\mathcal{B}-1} M[i \cdot C + j],$$

so that audited ballots are excluded automatically because their entries for M are forced to zero.

- **Voter Inclusion:**

Every non-audited ballot must be counted *exactly once*, and no extra ballots may be counted. This is enforced by the main election proof by requiring that for each ballot block $I_b = \{b \cdot C, \dots, b \cdot C + (C - 1)\}$, the mark column is well-formed and ballot-

consistent. This means for all $M[i] \in \{0, 1\}$, each ballot has at most one marked position, and audited ballots have no marked positions. Along with the global sum and tally constraints, this ensures that each cast ballot contributes exactly *once*, for one candidate only.

- **Dispute Resolution:**

After casting a ballot, the voter receives a receipt containing a confirmation code c associated with their ballot. Zeeperio supports two *separately verifiable* dispute proofs that match the two dispute questions: "Was my ballot included?" and "Is my confirmation code correct?". For the former scenario, the EA proves that the voter's code c is associated with a ballot \mathcal{B} and corresponds to a marked position in the committed election data. This is all done while hiding which candidate was selected. For the latter scenario, if the voter alleges that their receipt code is wrong, the EA proves *non-membership*. This means that c does not appear anywhere among the C confirmation codes printed on their ballot \mathcal{B} , without revealing any of those codes.

Zeeperio proves these properties with three separate proofs:

- **Main Election Proof:** Proves election-wide correctness (mark constraints, audit constraints, and tally constraints).
- **Ballot Inclusion Proof:** Proves voter ballot inclusion in a receipt-free manner.
- **Receipt Correctness Proof:** Proves non-membership (receipt correctness) for a claimed code.

6.1.1 Security Assumptions

Our security proof uses a standard cryptographic approach. We assume an adversarial prover who attempts to convince the verifier of a false statement and show that this leads to failure, thus being infeasible. We assume the field size satisfies $q \geq 2^\lambda$, where λ is the

security parameter ($\lambda \approx 100$ for the curve BN254). Zeeperio’s security relies on the KZG scheme’s *binding* property (knowledge soundness) and the Fiat-Shamir transformation for non-interactive challenges. We assume that the hash function used for Fiat-Shamir can be modeled as a random oracle, in which case the Fiat-Shamir transformation is proven to produce a secure non-interactive argument. We also assume the structured reference string (SRS) used in KZG is generated honestly, so that no prover knows the secret trapdoor. Under this assumption, the KZG commitment is binding and *no adversary can forge a valid proof without solving a hard discrete logarithm problem*. These assumptions are in line with standard SNARK security proofs [5]. If an adversary could produce a proof that satisfies all Zeeperio verification checks while the election is invalid, we can say they have broken these assumptions (thereby contradicting already proven security).

6.1.2 Soundness: Zeeperio Constraints \rightarrow Eperio Conditions

We first prove *encoding soundness*, where any accepted Zeeperio proof implies the election data must satisfy the Eperio conditions.

Theorem 6.1.1 (Encoding Main Proof Soundness). *Let π_{main} be the Zeeperio main election proof for a public instance*

$$x_{\text{main}} := (\text{election parameters}, \{\text{Tally}_c\}_{c=0}^{C-1}, \text{commitments to witness polynomials}).$$

If $\text{verifyMain}(x_{\text{main}}, \pi_{\text{main}}) = 1$, then except with negligible probability, the committed election columns satisfy Eperio’s validity conditions for: (i) audit/mark consistency, (ii) exactly-one-mark on cast ballots and zero marks on audited ballots, and (iii) tally correctness $\forall c \in \{0, \dots, C - 1\}$.

Proof.

We fix a PPT adversary \mathcal{A} that outputs a pair $(x_{\text{main}}, \pi_{\text{main}})$ such that $\text{verifyMain}(x_{\text{main}}, \pi_{\text{main}}) =$

1.

Step 1 (Existence of a consistent polynomial witness). If the main proof is valid, it implies that the polynomials committed inside the proof are *consistent* with the commitments and openings checked by the verifier. Because of KZG’s binding property, the openings in π_{main} uniquely fix the values of each committed polynomial at the sampled challenge point. Additionally, by knowledge soundness there exists (low-degree) witness polynomials that hold the verifier’s identities at the sampled challenge point.

Step 2 (From a satisfied random-point check to vanishing on Ω). In the main election proof, the verifier checks that a single batched polynomial identity holds at a fresh Fiat-Shamir challenge point $\zeta \xleftarrow{\$} \mathbb{F}_q$:

$$P_{\text{vanish}}(\zeta) = Q(\zeta) \cdot Z_{\Omega}(\zeta),$$

where $Z_{\Omega}(X)$ is the vanishing polynomial of the domain Ω , $P_{\text{vanish}}(X)$ aggregates all election constraints, and $Q(X)$ is the quotient polynomial.

We define the *error polynomial*:

$$E(X) := P_{\text{vanish}}(X) - Q(X) \cdot Z_{\Omega}(X).$$

$E(X)$ has degree $< d$ for some explicit bound $d \ll q$. If the verifier accepts, this implies $E(\zeta) = 0$. If $E(X)$ is the zero polynomial, then $P_{\text{vanish}}(X) = Q(X) \cdot Z_{\Omega}(X)$ holds, which implies $P_{\text{vanish}}(X)$ vanishes on every $X \in \Omega$ because $Z_{\Omega}(X) = 0$ for all $X \in \Omega$. If $E(X)$ is nonzero, then by the Schwartz-Zippel lemma:

$$\Pr_{\zeta \xleftarrow{\$} \mathbb{F}_q} [E(\zeta) = 0] \leq \frac{d}{q} \leq \frac{d}{2^{\lambda}} = \text{negl}(\lambda),$$

for any d bounded by a polynomial in λ . Thus, except with negligible probability, we are in the first case where P_{vanish} vanishes on Ω , and *all aggregated election constraints hold on*

Ω .

Step 3 (Election constraints imply Eperio validity). Because P_{vanish} batches the individual constraints, $P_{\text{vanish}}(X) = 0$ for all $X \in \Omega$ implies each constraint holds on its intended indices as follows.

Mark/Audit Conditions per Ballot. For each ballot \mathcal{B} and each $i \in I_b$, the constraints enforce that $A[i] \in \{0, 1\}$, $M[i] \in \{0, 1\}$ and that the audit status is consistent across the block:

$$(\forall i \in I_b : A[i] = 1) \rightarrow (\forall i \in I_b : M[i] = 0),$$

, and

$$(\forall i \in I_b : A[i] = 0) \rightarrow \sum_{i \in I_b} M[i] = 1.$$

Thus, every audited ballot has no counted mark value, and every cast ballot has exactly one mark. This is exactly as in Eperio, where the ballots are well-formed (no overvotes/undervotes among cast ballots) and the inclusion condition holds (audited ballots are excluded).

Tally Correctness. We fix a candidate position $c \in \{0, \dots, C - 1\}$. Consider the *stream* of indices

$$S_c := \{c, c + C, c + 2C, \dots\},$$

which correspond to the c th candidate position on every ballot. The tally check is enforced using a strided accumulator polynomial $\text{Acc}_M(X)$ with stride ω^C :

$$\text{Acc}_M(\omega^i) - M(\omega^i) + \text{Acc}_M(\omega^{i+C}) = 0 \quad \text{for the stream recurrence,}$$

$$\text{Acc}_M(\omega^i) - M(\omega^i) = 0 \quad \text{at the last } C \text{ positions,}$$

$$\text{Acc}_M(\omega^c) - \text{Tally}_c = 0 \quad \text{at the first position.}$$

We now show this implies $\text{Tally}_c = \sum_{j=0}^{\mathcal{B}-1} M[j, c]$. Let $i_0 := c$ and $i_{t+1} := i_t + C \pmod{n}$ be

the stream positions. For the stream recurrence, and for each t before the last position,

$$\text{Acc}_M(\omega^{i_t}) = M(\omega^{i_t}) - \text{Acc}_M(\omega^{i_{t+1}}).$$

By doing the recurrence until the last constraint applies gives us:

$$\text{Acc}_M(\omega^{i_0}) = \sum_{t=0}^{L-1} M(\omega^{i_t}),$$

where L is the stream length (including padded zeros, which do not change the sum).

Finally, the starting constraint sets $\text{Acc}_M(\omega^{i_0}) = \text{Tally}_c$, hence

$$\text{Tally}_c = \sum_{t=0}^{L-1} M(\omega^{i_t}) = \sum_{j=0}^{\mathcal{B}-1} M[j, c],$$

which gives us tally correctness for candidate c .

By combining both substeps, the accepted main proof implies the election record is valid in relation to Eperio. Every cast ballot contributes exactly one vote to exactly one candidate stream, audited ballots contribute none, and the announced tallies equal the per-candidate sums. □

Theorem 6.1.2 (Ballot inclusion soundness). *We assume the main proof has been accepted for the committed election columns. Let b be a ballot ID and c a confirmation code provided by a voter. If $\text{verifyInclusion}(b, c, \pi_{\text{incl}}) = 1$, then except with negligible probability, there exists an index $i \in I_b$ such that $M[i] = 1$ and $C_{\text{confirm}}[i] = c$ in the committed election data. This means ballot b was included as a cast ballot in the tally.*

Proof.

The inclusion proof is built over the committed *shuffled* columns $\text{BallotID}_\pi(X)$, $\text{Mark}_\pi(X)$, and $\text{Code}_\pi(X)$ for which there exists a permutation π that is applied to the original columns.

In the query phase, the prover supplies an index t and KZG openings at the point $X = \omega^t$.

If the verifier accepts the proof, it implies:

$$\text{BallotID}_\pi(\omega^t) = b,$$

$$\text{Mark}_\pi(\omega^t) = 1,$$

$$\text{Code}_\pi(\omega^t) = c,$$

and the KZG opening proof verifies for these evaluations. By KZG binding, these equalities are true evaluations of the committed polynomials at ω^t .

We need to relate the shuffled columns back to the original election columns. The inclusion proof proves the permutation argument that the shuffled triple $(\text{BallotID}_\pi, \text{Code}_\pi, \text{Mark}_\pi)$ is obtained from $(\text{BallotID}, \text{Code}, \text{Mark})$, by applying a single permutation π to the rows. Thus, soundness of the permutation argument implies there exists a permutation π such that for every row index u ,

$$(\text{BallotID}_\pi(\omega^u), \text{Code}_\pi(\omega^u), \text{Mark}_\pi(\omega^u)) = (\text{BallotID}(\omega^{\pi(u)}), \text{Code}(\omega^{\pi(u)}), \text{Mark}(\omega^{\pi(u)})).$$

We instantiate $u = t$ and let $i := \pi(t)$. Then:

$$\text{BallotID}(\omega^i) = b, \quad \text{Mark}(\omega^i) = 1, \quad \text{Code}(\omega^i) = c.$$

By the definition of the ballot ID column, $\text{BallotID}(\omega^i) = b$ implies $i \in I_b$ (ignoring padding, which has ballot ID = 0). Hence there exists $i \in I_b$ with $M[i] = 1$ and $C_{\text{confirm}}[i] = c$.

Finally, under the accepted main election proof, any ballot block with a marked position must be a cast (non-audited) ballot and thus included in the tally. Therefore, ballot b was included as cast. □

Theorem 6.1.3 (Receipt Proof Soundness (Non-Membership)). *We assume the main proof has been accepted for the committed election columns. Let b be a ballot ID and $c \in \mathbb{F}_q$ a disputed confirmation code claimed by a voter. If $\text{verifyReceipt}(b, c, \pi_{\text{rcpt}}) = 1$, then except with negligible probability, c does not appear among the C confirmation codes for ballot b in the committed election data:*

$$\forall i \in I_b : C_{\text{confirm}}[i] \neq c.$$

Proof.

The receipt proof encodes a non-membership statement using a ballot selector mask and an *inverse witness* polynomial. Let $\text{Mask}_{\text{ballot}}^b(X)$ be the public 0/1 mask that equals 1 on the ballot block I_b and 0 elsewhere on Ω . Let $\text{Inv}_b(X)$ be an auxiliary witness polynomial introduced by the prover. We define the dispute-check polynomial

$$V_{\text{dispute}}(X) := \left((\text{Code}(X) - c) \cdot \text{Inv}_b(X) - 1 \right) \cdot \text{Mask}_{\text{ballot}}^b(X).$$

If the verifier accepts, this implies that by the Schwartz-Zippel lemma, $V_{\text{dispute}}(X) = 0$ for all $X \in \Omega$, except with negligible probability.

We fix $i \in I_b$. Then $\text{Mask}_{\text{ballot}}^b(\omega^i) = 1$, so

$$0 = V_{\text{dispute}}(\omega^i) = (\text{Code}(\omega^i) - c) \cdot \text{Inv}_b(\omega^i) - 1.$$

Rearranging gives

$$(\text{Code}(\omega^i) - c) \cdot \text{Inv}_b(\omega^i) = 1.$$

In a field, a product equals 1 only if both factors are nonzero. If $\text{Code}(\omega^i) - c \neq 0$, then $\text{Code}(\omega^i) \neq c$. Since this holds for every $i \in I_b$, we conclude that c does not appear among the C confirmation codes for ballot b .

Therefore, if the voter claims c is their received code, an accepting receipt proof is evidence that the disputed code is not associated with ballot b . \square

It is important to note that if any of the above properties were violated by the EA, building an accepting proof π would require solving the discrete log problem, or predicting the Fiat-Shamir challenges. For example, if the tallies were wrong or a ballot was not counted properly, the polynomial $Q(X)$ that encapsulates all constraints would not vanish at the random challenge point ζ as expected. Therefore, the verifier’s pairing check would fail with overwhelming probability by the Schwartz-Zippel lemma [113].

An adversary that builds polynomial openings to hide this would be forging a KZG proof, which is infeasible without the secret trapdoor or solving discrete log problems. Therefore, we conclude that no adversary can produce an accepting Zeeperio proof for an election that violates tally correctness, voter inclusion, or dispute resolution.

6.1.3 Completeness: Eperio Conditions \rightarrow Existence of Proof

We now prove *encoding completeness*, where if the election is conducted correctly, satisfying all of Eperio’s conditions, then an honest prover can always generate a valid Zeeperio proof.

Theorem 6.1.4 (Main proof completeness). *Assuming the EA follows the protocol and the election instance satisfies Eperio validity, then there exists a Zeeperio main proof π_{main} such that*

$$\text{verifyMain}(x_{\text{main}}, \pi_{\text{main}}) = 1.$$

Proof. We assume the election transcript is valid in relation to Eperio. We construct explicit witness polynomials that satisfy every main proof constraint on the entire domain Ω .

Step 1 (Define Evaluation Vectors). We build length- n vectors (with zero padding on indices $i \geq \mathcal{P}$):

- $A[i] \in \{0, 1\}$ the audit indicator, constant on each ballot block I_b ;
- $M[i] \in \{0, 1\}$ the mark indicator;
- $C_{\text{confirm}}[i] \in \mathbb{F}_q$ the confirmation code column.

Step 2 (Interpolation). We interpolate these vectors over Ω to obtain the unique degree $< n$ polynomials $\text{Audit}(X)$, $\text{Mark}(X)$, and $\text{Code}(X)$ such that for all $i \in \{0, \dots, n-1\}$:

$$\text{Audit}(\omega^i) = A[i], \quad \text{Mark}(\omega^i) = M[i], \quad \text{Code}(\omega^i) = C_{\text{confirm}}[i].$$

Step 3 (Build Auxiliary Polynomials for each Constraint). For every booleanity constraint on a column $F \in \{\text{Audit}, \text{Mark}\}$, define evaluations

$$V_F(\omega^i) := F(\omega^i) \cdot (F(\omega^i) - 1),$$

and interpolate $V_F(X)$. Since $A[i], M[i] \in \{0, 1\}$, we have $V_F(\omega^i) = 0$ for all i .

For per-ballot constraints, define the block-sums on each ballot block:

$$S_{\text{blk}}^M(b) := \sum_{t=0}^{C-1} M[b \cdot C + t], \quad S_{\text{blk}}^A(b) := \sum_{t=0}^{C-1} A[b \cdot C + t],$$

and interpolate the corresponding block-sum polynomials.

Eperio validity implies if ballot b is audited, then $S_{\text{blk}}^M(b) = 0$. If ballot b is cast (not audited), then $S_{\text{blk}}^M(b) = 1$. Additionally, $A[i]$ is constant across I_b . Hence all ballot-block constraints hold on Ω .

For tally correctness, for each candidate stream $c \in \{0, \dots, C-1\}$, we define the strided

accumulator evaluations:

$$\text{Acc}_{M_c}(\omega^{c+j \cdot C}) := \sum_{\ell=j}^{\mathcal{B}-1} M[c + \ell C],$$

and $\text{Acc}_{M_c}(\omega^i) := 0$ on padded indices. We interpolate to get $\text{Acc}_{M_c}(X)$. Then the standard strided recurrence and starting position constraint hold by construction:

$$\text{Acc}_{M_c}(\omega^{c+j \cdot C}) = M[c + j \cdot C] + \text{Acc}_{M_c}(\omega^{c+(j+1) \cdot C}) \quad \text{and} \quad \text{Acc}_{M_c}(\omega^c) = \text{Tally}_c,$$

where $\text{Tally}_c = \sum_{j=0}^{\mathcal{B}-1} M[c + j \cdot C]$ holds by Eperio tally correctness.

Step 4 (Satisfy Global Vanishing Identity). Let $P_{\text{vanish}}(X)$ be the protocol’s batched constraint polynomial built from the polynomials above, and let $Z_{\Omega}(X)$ be the vanishing polynomial of Ω . Since each constraint is satisfied on Ω , it follows that $P_{\text{vanish}}(\omega^i) = 0$ for all $\omega^i \in \Omega$. We define the quotient polynomial $Q(X)$ by the identity

$$P_{\text{vanish}}(X) = Q(X) \cdot Z_{\Omega}(X),$$

which exists as a polynomial because Z_{Ω} divides P_{vanish} (it has roots at all points of Ω). **Step 5 (Commit and open \rightarrow Verifier Accepts).** The EA computes KZG commitments to all required witness polynomials, derives the Fiat-Shamir challenges deterministically from the transcript, and outputs the required openings at the challenge points along with valid KZG opening proofs. Because all identities hold as polynomial identities (not probabilistically), the verifier’s algebraic checks hold exactly, and the KZG verification equations accept. Therefore $\text{verifyMain}(x_{\text{main}}, \pi_{\text{main}}) = 1$. □

Theorem 6.1.5 (Ballot Inclusion Completeness). *We assume the election instance is valid in relation to Eperio, and the main proof has been generated from the committed election data. Let b be a ballot ID and let c be the confirmation code on the voter’s receipt. Then*

there exists a Zeeperio inclusion proof π_{incl} such that

$$\text{verifyInclusion}(b, c, \pi_{\text{incl}}) = 1.$$

Proof.

Because the election is proven to be valid in relation to Eperio, ballot b is either audited, or cast. The theorem concerns the case where the voter presents a valid receipt for the cast ballot, so there exists an index $i^* \in I_b$ such that

$$M[i^*] = 1 \quad \text{and} \quad C_{\text{confirm}}[i^*] = c.$$

The inclusion prover builds the proof witness as follows.

Step 1 (Shuffled Columns). Let $\text{BallotID}(X)$ be the (public) ballot-id column polynomial and let $\text{Mark}(X), \text{Code}(X)$ be the committed mark and code polynomials. We define a permutation π of the n rows that places row i^* at some chosen position t . We define shuffled evaluation vectors:

$$\text{BallotID}_\pi[u] := \text{BallotID}[\pi(u)], \quad M_\pi[u] := M[\pi(u)], \quad C_{\text{confirm},\pi}[u] := C_{\text{confirm}}[\pi(u)],$$

and interpolate to get the shuffled polynomials $\text{BallotID}_\pi(X), \text{Mark}_\pi(X), \text{Code}_\pi(X)$.

Step 2 (Permutation Argument Witness). Using the same permutation construction as in Chapter 5, the prover builds the auxiliary witness polynomials that certify that the triple $(\text{BallotID}_\pi, \text{Mark}_\pi, \text{Code}_\pi)$ is a permutation of $(\text{BallotID}, \text{Mark}, \text{Code})$.

Step 3 (Opening Target Row). By construction of π and the choice of t with $\pi(t) = i^*$, the shuffled polynomials satisfy:

$$\text{BallotID}_\pi(\omega^t) = b, \quad \text{Mark}_\pi(\omega^t) = 1, \quad \text{Code}_\pi(\omega^t) = c.$$

The prover outputs the KZG openings attesting to these evaluations, along with the openings needed by the permutation check.

Step 4 (Verifier accepts). All proof constraints are satisfied. The permutation constraints are satisfied by construction of π and its grand-product witness, and the target row equality constraints hold because row t is exactly the permuted version of i^* . Therefore the verifier's checks pass and $\text{verifyInclusion}(b, c, \pi_{\text{incl}}) = 1$. \square

Theorem 6.1.6 (Receipt Proof Completeness (Non-Membership)). *We assume the election instance is valid in relation to Eperio, and the main proof has been generated from the committed election data. Let b be a ballot ID and let c' be a disputed confirmation code such that c' does not appear among the C confirmation codes on ballot b :*

$$\forall i \in I_b : C_{\text{confirm}}[i] \neq c'.$$

Then there exists a Zeeperio receipt proof π_{rcpt} such that

$$\text{verifyReceipt}(b, c', \pi_{\text{rcpt}}) = 1.$$

Proof.

Let $\text{Mask}_{\text{ballot}}^b(X)$ be the public ballot selector polynomial that equals 1 on indices in I_b and 0 elsewhere on Ω .

Step 1 (Define Inverse Witness Evaluations). We define the evaluations of an auxiliary polynomial $\text{Inv}_b(X)$ on Ω by:

$$\text{Inv}_b(\omega^i) := \begin{cases} (\text{Code}(\omega^i) - c')^{-1} & \text{if } i \in I_b, \\ 0 & \text{if } i \notin I_b. \end{cases}$$

These inverses exist for $i \in I_b$ because $\text{Code}(\omega^i) = C_{\text{confirm}}[i] \neq c'$ by assumption. We

interpolate these values to obtain $\text{Inv}_b(X)$ of degree $< n$.

Step 2 (Show Dispute Polynomial Vanishes on Ω). We consider the dispute-check vanishing polynomial:

$$V_{\text{dispute}}(X) := \left((\text{Code}(X) - c') \cdot \text{Inv}_b(X) - 1 \right) \cdot \text{Mask}_{\text{ballot}}^b(X).$$

For $i \in I_b$, $\text{Mask}_{\text{ballot}}^b(\omega^i) = 1$ and by construction $(\text{Code}(\omega^i) - c') \cdot \text{Inv}_b(\omega^i) = 1$, so $V_{\text{dispute}}(\omega^i) = 0$. For $i \notin I_b$, $\text{Mask}_{\text{ballot}}^b(\omega^i) = 0$, so $V_{\text{dispute}}(\omega^i) = 0$. Hence V_{dispute} vanishes on all of Ω .

Step 3 (Commit and open \rightarrow Verifier Accepts). The prover commits to the required polynomials including Inv_b , the quotient polynomial, derives the Fiat-Shamir challenges, and outputs valid KZG openings at the verifier’s challenge points. Since the defining identities hold exactly on Ω , the verifier’s check at the challenge point holds and the KZG opening verification succeeds. Therefore $\text{verifyReceipt}(b, c', \pi_{\text{rcpt}}) = 1$. \square

These proofs show that Zeeperio’s constraints are faithful to Eperio. They are both sound and complete with respect to Eperio’s tally correctness, voter inclusion, and dispute resolution properties.

6.2 Implementation

We implemented Zeeperio with two main components, an off-chain prover and an on-chain verifier smart contract. We show how the prover code generates the three required proofs and how it optimizes the proof data for efficient on-chain cost. We then describe the verifier contract, which uses Ethereum precompiled cryptographic operations to verify Zeeperio proofs on-chain.

6.2.1 Prover Overview

The Zeeperio prover is implemented in Rust, using the *Arkworks* library for elliptic curve operations and polynomial commitments. The prover is designed as a command-line tool (CLI) that takes as input the election parameters and data, and it outputs the corresponding proofs. The prover codebase is organized into different modules, with each one covering different phases of the Poly-IOP protocol. The *Arkworks* library supports the same curve used in Ethereum’s precompiles for BN254, introduced in EIPs 196 and 197 [137, 138]. These precompiles allow for efficient group operations and pairing computations on the BN254 curve.

CLI and Proof Generation

The CLI requires the EA to input the election parameters, which include: the number of audited ballots, the number of cast ballots, and the claimed tally results. The program can then generate the three proofs:

- **Main election proof:** This proof shows the correctness of the tally, and the overall consistency of the election record.
- **Ballot inclusion proof:** This proof shows that the shuffled ballots are indeed a permutation of the cast ballots. This allows voters to later verify that their ballot is included in the tally.
- **Receipt correctness proof:** This proof shows that given a fake receipt, there is no valid ballot in the election record that corresponds to it.

In an election deployment scenario, the ballot inclusion proof is generated, however, it is only used in the event a voter wants verification that their ballot is included in the tally. Similarly, the receipt correctness proof is only generated and used if a voter does not believe their receipt corresponds to their ballot. The proof generation process follows the Poly-IOP

protocol as described in Chapter 5:

1. Polynomial Representation:

The prover first represents each constraint as a polynomial equation. It builds the mark polynomial M , audit polynomial A , and the confirmation code polynomial C_{confirm} , each of degree $n - 1$ (where $n = C \cdot \mathcal{B} + \text{padding}$). It also builds the derived polynomials, such as the *block-sum* polynomial using M , A in blocks of size C , and the *tally accumulator* polynomial that encodes the running sum of each candidate. At the end, the *quotient* polynomial Q is built to encapsulate all enforced constraints, including the *vanishing* polynomial described in Chapter 5.

2. Polynomial Commitments:

The prover then commits to each of these polynomials using the KZG commitment scheme. Each polynomial $F(X)$ that is committed uses a curve point $[F] = g_1^{F(\tau)}$, where τ is the secret trapdoor from the SRS and g_1 is a generator of the G_1 group. To enforce zero knowledge, the prover adds a random blinding factor r , resulting in the commitment $[F] = g_1^{F(\tau)} \cdot h^r$, where h is another generator of the G_1 group. Each polynomial commitment is constant-sized, where the commitment is just one elliptic curve point.

3. Non-Interactive Proof Generation:

After the commitments, the prover uses the *Fiat-Shamir* heuristic to generate random challenges to combine and evaluate the committed polynomials. In our implementation, these challenges are derived from the transcript, which includes the concatenation of the binary encodings of all commitments and public inputs. This is then hashed using the Keccak-256 cryptographic hash function, and the result is represented as a field element to obtain ζ . This process emulates the random oracle, ensuring that the derived challenges are unpredictable to the prover. For the verifier to correctly *verify* these proofs, both the prover and the verifier must use consistent

serialization for this hash. In our implementation, we carefully define ordering and byte format so that the verifier smart contract can reproduce the exact hash and get the identical ζ .

4. **Random Linear Combination:**

Given the challenge ζ , the prover computes the openings of the committed polynomials at $X = \zeta$. Rather than sending the entire polynomial evaluations separately, Zeeperio uses an optimization, by performing a batched opening using *random linear combination*. Instead of producing separate KZG proofs $\pi_F = \text{KZG.Open}(F, \zeta)$ for each polynomial F , the prover computes one aggregated proof π_{batched} such that the verifier can check a single pairing equation to simultaneously verify $F(\zeta) = y_F$ for multiple polynomials F . The prover picks another random scalar γ (via Fiat-Shamir) and computes a single combined proof that attests to all polynomial evaluations at ζ at once. In our implementation, the prover picks another random scalar γ (via Fiat-Shamir) to form a combined evaluation value, and produces **one** KZG opening proof π_{open} for the entire evaluation. This drastically reduces the number of pairings the verifier has to do on-chain, resulting in a lowered overall cost.

5. **Proof Output:**

Finally, the prover builds the proof outputs. For each proof (Main, Inclusion, Receipt), the output consists of the list of commitments to key polynomials, the claimed evaluation results of those polynomials at ζ , and the single combined opening proof. These are encoded according to Ethereum's *Application Binary Interface* (ABI) format as a byte array [139]. It is important to note that the proof sizes are constant. This means they do not grow with the number of ballots \mathcal{B} , which is one of the main benefits of such zk-SNARKs.

Calldata Optimization

One major aspect of our implementation is how the proofs are optimized for Ethereum smart contracts. Pushing data to a smart contract costs gas (unit of work) proportional to the data size, so keeping proofs small is very important. As mentioned before, using KZG commitments keeps each commitment constant-size (32 bytes compressed) regardless of the polynomial length, and batching reduces the number of separate proofs. We also ensure that the proof is packed tightly. The ABI encoding concatenates all field elements and curve points. Since the verifier contract does not need to store anything long-term (it just performs calculations and returns a boolean), all proof data is passed in calldata, which is cheaper than writing to storage [140].

6.2.2 Verifier Overview

The Zeeperio verifier is implemented in a Solidity smart contract. This contract was developed and tested using Foundry, which is an Ethereum development toolkit. Foundry provides a framework for writing Solidity code, deploying it to local and test networks, and performing unit tests, which we used extensively to ensure correctness before deployment [141].

Verifier Smart Contract Design

The smart contract, written in Solidity, includes three public functions: *verifyMain*, *verifyInclusion*, and *verifyReceipt*, corresponding to the three proofs generated by the prover. Each function takes as input the ABI-encoded proof, and outputs a boolean indicating whether the proof is valid. We separate the functions for convenience and for modularity, even though they share a lot of helper logic.

Inputs and Parsing

When a proof is submitted via the CLI as a transaction calling one of these functions, the contract first parses the bytes to extract the structured data, including the public inputs (like tallies), the commitments, and the opening proof. Solidity does not natively support elliptic curve point types, so we represent the x and y coordinates as pairs of `uint256`. The contract includes methods for slicing the input bytes and interpreting 32-byte chunks as `uint256` values. The inputs are verified implicitly by using them with Ethereum’s precompiled operations. If the points are invalid, the precompiles will fail.

Recomputing Challenges

Next, the verifier recomputes the Fiat-Shamir challenge values exactly as the prover did. This is a very important step as it ensures the proof’s internal consistency. For example, in the function `verifyMain`, the contract hashes the concatenation of all commitments and public inputs to derive the challenge ζ . We use Ethereum’s `keccak256` hash function (the default Solidity hash) as the random oracle. The serialization (byte order of each field element) was carefully defined so that the hash in Solidity yields the identical ζ that the prover used. This step binds the proof components together because a malicious prover cannot tamper with a commitment or value without changing the hash. This would ultimately change the expected challenge, causing the verification to fail.

Pairing Checks via Precompiles

Once the challenges are set, the verifier performs the core verification of checking the KZG opening proof using Ethereum’s elliptic curve pairings. Ethereum provides precompiled contracts for operations on the BN254 curve where `BN256Pairing` (ETH address 0x8) allows us to do a pairing product check, and `BN256Add`/`BN256Mul` (ETH addresses 0x6 and 0x7) allows for point addition and scalar multiplication [137, 138]. It is important to note that, even though the precompiles are named BN256, they actually implement the

BN254 curve used in Ethereum. Our contract uses these precompiles to implement the KZG.Verify function. Verifying a KZG opening requires the contract to check:

$$e(F^\tau - g_1^{y_F}, g_2) \stackrel{?}{=} e(\pi_F, g_2^{\tau-\zeta})$$

where F^τ is the commitment in G_1 , g_2 is the generator of G_2 , ζ is the challenge, τ is from the SRS, y_F is the claimed evaluation at ζ , and π_F is the opening proof in G_1 .

In our implementation, this equation is rearranged and fed into the precompile as a single pairing product verification. For the batched proofs, y is a combination of the multiple polynomial evaluations. Our contract builds the necessary G_1 and G_2 points using *BN256Add* to combine points and multiplying the G_2 generator by ζ using *BN256Mul* to get g_2^ζ . Then it calls the *BN256Pairing* precompile with the appropriate pairs. If the pairing check returns true (success), the proof is accepted; otherwise it is rejected. All of this is done with one pairing check, as it can handle multiple pairings in one call by aggregating the input points.

The verification logic for each proof type is slightly different due to the commitments and equations being different, but they all share the same pattern. The verifier recomputes the challenges and performs one, single pairing check. The Main proof verification takes the commitments, computes ζ , and then verifies the vanishing polynomial equation $P_{\text{vanish}}(\zeta) - Q(\zeta) \cdot Z_H(\zeta) = 0$ (as described in Chapter 5) with one pairing. The Inclusion proof and Receipt proof verification takes the same commitments, along with commitments to the ballot IDs and confirmation codes as inputs, and uses a pairing check to ensure the commitments open to the proper code values.

Foundry

We used Foundry to write unit tests that simulate various scenarios. For example, we tested that a valid proof generated by the Rust prover for a sample election returns *true*, while

altering any byte, like changing a tally or a commitment slightly, causes verification to return *false*. Foundry allows us to automate feeding in large proof data and measuring the execution.

Sepolia Testnet

The verifier contract was deployed to the *Sepolia* testnet for E2E integration. We chose Sepolia because it supports the BN254 precompiles and has a similar environment to Ethereum mainnet [142]. We then ran a live test, simulating the transactions for a sample 100,000 ballot election. All transactions returned true, confirming that the smart contract can handle the proof and that the gas usage is within acceptable limits. The on-chain results confirmed that our calldata optimizations and batching significantly reduced the overall cost. If we had verified each polynomial separately, that would have required many more pairings and thus, a much higher gas cost.

6.3 Results and Discussion

We evaluated Zeeperio on an election with 100,000 ballots and 5 candidates, reflecting a large-scale, real-world scenario. The details of the sample election are shown in Table 6.1. The system successfully generated proofs for this entire election and verified them on-chain.

Table 6.1: Fixed sample election instance used for benchmarking.

Parameter	Value
Ballots (\mathcal{B})	100,000
Candidates (C)	5
Positions ($\mathcal{P} = \mathcal{BC}$)	500,000
Padding (n)	$2^{20} = 1,048,576$
Print-audit allocation	10%
Audited ballots ($\mathcal{B}_{\text{audit}}$)	10,000

Performance

As shown in Table 6.2, the Zeeperio prover completed the main election proof in roughly 2.5 hours on a single core of a MacBook Pro (Apple M3 Pro chip, 18 GB RAM). Additionally, it generated the inclusion proof and receipt proof in under an hour. In total, producing all proofs took under 5 hours of single core computation, while they can be done simultaneously for under 3 hours. Regarding the proof sizes, the main election proof takes ~ 8.4 KB of calldata, while the inclusion and receipt proofs are smaller at ~ 4.1 KB and ~ 2.7 KB respectively. This is orders of magnitude faster than other comparable SNARKs and STARKs in recent literature.

Table 6.2: Zeeperio benchmark with sample election. All measurements are from a single-core execution on standard hardware (MacBook M3 Pro).

Metric	Main	Inclusion	Receipt
Proof generation (s)	8,709	3,130	3,128
Commitments (#)	24	13	6
Opening points (#)	6	2	2
Calldata size (bytes)	8,416	4,128	2,688
Gas used	2,429,116	1,029,029	727,118

On Ethereum Sepolia testnet, verifying Zeeperio’s proofs took ~ 4.2 million gas in total, which translates to $\sim \$30$ USD in transaction fees.¹ The gas cost is mainly because of the cryptographic verification rather than data transfer, thanks to the small proof sizes. For context, 4.2 million gas is roughly 7% of the block gas limit on Ethereum (~ 60 million gas), so the verification easily fits in a single block, even with other transactions present.

Mainnet Feasibility

A \$30 on-chain verification cost for a 100,000 ballot election is very reasonable for a real-world deployment. Even if gas prices or ETH prices fluctuate, the cost is on the order of tens to a few hundreds of dollars, which is negligible in an official election. For example, if Ethereum gas prices are 10x higher (gas price of 20 Gwei), the total cost would be around \$250 USD, which is still affordable for election authorities. Therefore, deploying Zeeperio on Ethereum mainnet to verify real election results is very practical. This brings us closer to *autonomous* election verification, where *someone* actually verifies the proofs. The on-chain verification is also very fast, taking only a few seconds. This means results can be confirmed to the public almost immediately after the proofs are posted.

Comparison with other SNARKs and STARKs

We compare Zeeperio’s results with the different alternatives in the literature. We compare against a general-purpose (GP) Groth16 zk-SNARK-based approach by Huber et al., and a zk-STARK-based approach by Harrison & Haines [6, 143].

Table 6.3 provides a high-level comparison. It should be noted these comparisons are not one-to-one comparisons, as they each prove different statements. While Zeeperio proves correctness for Eperio style elections, the GP SNARK proves ballot validity (each ballot is well-formed, more related to the *cast-as-intended* property), and the STARK focuses on *counted-as-collected* property in homomorphic tallies. Although these prove different

¹This is assuming a gas price of 2 Gwei, and an ETH price of \$3000 USD, while 1 Gwei = 10^{-9} ETH.

statements, they are all approaches for E2E verifiable elections using succinct proofs, so the comparison is still valuable. Zeeperio's custom SNARK is highly efficient but specialized to the Eperio-style election. However, this is a design by choice, as Zeeperio's constraints can be modified to support other election types as well. We show by focusing on a *specific* election type, we can achieve much better performance than general-purpose systems.

Table 6.3: Zeeperio compared against per-ballot GP-SNARK and ZK-STARK approaches.

Protocol / Approach	Proof Size	Prover Time	Verification Cost (Gas)	Trusted Setup	Post-Quantum?
Zeeperio	~8.4 KB (main proof); ~15 KB total for all proofs	~ under 5 hours single-core	~4.2 million gas	Yes (one-time SRS, reused)	No (elliptic curve pairings)
GP SNARK	~192 bytes per ballot proof (constant) (~20 MB for 100k ballots)	~0.05 s per simple ballot (~1 hour 30 minutes for 100k ballots)	Each proof requiring ~100k gas to verify (~10 billion gas for 100k proofs, impractical without batching)	Yes (per-circuit or universal SRS)	No (elliptic curve pairings)
ZK-STARK	Proof in megabytes for 10s of thousands of ballots (grows polylogarithmically with ballots)	~223 hours for 32k ballots (on a single core)	Millions of gas (hash heavy); No on-chain example at 100k (verification grows with proof size)	No (transparent, no setup)	Yes (uses hashes)

From Table 6.3, we analyze the main differences:

- **Trusted Setup:**

Both Zeeperio and the Groth16-based proofs require an SRS. Zeeperio uses a universal *Powers of Tau* (PoT) ceremony which is one-time (in our implementation, we reused Ethereum’s publicly-contributed 2^{28} PoT, avoiding needing a new ceremony). This means any number of elections up to that size can use the same SRS securely. The GP SNARK approach could similarly use a universal SRS, but it requires another ceremony on a per-circuit basis. On the other hand, zk-STARKs are *transparent* (no trusted setup) which is the main advantage in terms of trust assumptions. However, since a widely trusted PoT already exists, this is not really a factor as Ethereum performed the ceremony with over 140,000 participants [144]. If only *one* of those participants deleted their randomness, the SRS is secure.

- **Post-Quantum Security:**

Zk-STARKs are built on cryptographic hash functions and are considered post-quantum secure. Zeeperio uses the BN254 pairings, which is not post-quantum secure. There is a potential scenario where a large quantum computer that could break discrete logs would compromise its security. However, given the current state of quantum computing, this risk is largely theoretical for the foreseeable future. Like Zeeperio, the GP SNARK (Groth16) is not post-quantum secure because it also relies on elliptic curve pairings. If post-quantum readiness is a priority (looking ahead a decade or more), a STARK or other post-quantum proof system might be necessary. There is ongoing research into post-quantum SNARKs, but none are as efficient as current pairing-based systems. Zeeperio chooses performance over post-quantum security, but since Zeeperio is a custom SNARK, we can switch the KZG commitment scheme for a post-quantum alternative in the future if needed.

- **Proof Succinctness:**

Zeeperio’s proofs are *highly succinct* because the proof size is independent of the number of ballots ($O(1)$ in complexity). The GP SNARK approach also has constant-

size proofs per ballot, but if we were to submit all proofs, the total data scales with number of ballots ($O(\mathcal{B})$ in complexity). One could batch many SNARK proofs using techniques like proof aggregation or recursive SNARKs, but that introduces additional complexity and overhead. While zk-STARKs are *theoretically* quasi-succinct (polylogarithmic size), they have large constants, so the proof size grows noticeably with the size of the problem. For 100,000 ballots, a zk-STARK proof can be on the order of several megabytes, which would be very expensive to verify on-chain. Therefore, Zeeperio is extremely efficient, which is very important for on-chain use as gas costs scale with proof size.

- **Prover Time:**

Zeeperio’s prover is significantly faster than the STARK prover for this specific application. The polynomial IOP approach was tailored to the election itself, avoiding additional overhead. Because a GP STARK (or even a GP SNARK) uses a circuit-based approach, it has to go through millions of constraints and is less optimized when compared to a *custom* SNARK for a specific problem. Zeeperio’s results show that one could scale to even larger elections with very reasonable computational time. The zk-STARK approach is impractical beyond a certain point, as the 32,000 ballots case already took ~ 9 days of CPU time. However, since the testing was done on a single core, parallelizing the prover could speed this up significantly. On the contrary, the GP SNARK is also very fast, so if the proofs can be aggregated, it is also feasible. One thing to note is that it lacks a single unified proof of the whole election, unless the votes are homomorphically tallied (which changes the SNARK design).

- **Verification Cost:**

Zeeperio’s on-chain verification is extremely cheap relative to the election size. It takes ~ 4.2 million gas for verifying an election with 100,000 ballots. The GP SNARK method shows proofs for each individual ballot. If one verified each proof

individually on-chain, that would cost ~100 thousand gas per ballot (verifying a Groth16 SNARK involves 2 pairing checks and some hashing). The total gas cost would be

$$100,000 \text{ ballots} \times 100,000 \text{ gas/ballot} = 10,000,000,000 \text{ gas,}$$

which would take over 30,000 blocks worth of gas and is impossible to fit in a single block on Ethereum. Obviously, one would not design it in this way. Instead, one would need to *aggregate* multiple zk-SNARK proofs to verify them together (there are schemes in the literature that verify multiple Groth16 proofs in one go, but those themselves become complex and expensive) [145]. Zk-STARK verification involves many hash computations (Merkle tree checks and FRI polynomial checks). Hashing is cheaper than pairings on Ethereum, but not free. Harrison & Haines did not publish the gas costs of verifying such proofs, however, a zk-STARK in the size of MBs likely costs in the order of millions of gas as well. The study suggested STARK verification for a similar voting context is manageable but would consume significant resources (their focus was more on theoretical feasibility).

As such, Zeeperio’s approach provides orders-of-magnitude improvements in proof size and proving time when compared to the GP zk-SNARK and zk-STARK approaches. The main trade-off is that we rely on a trusted setup with classical hardness assumptions, however, the benefits are substantial: Zeeperio can provide practical, routine, automated verification of large elections. On the contrary, if one’s threat model extends to quantum attackers or distrust of any setup, then one might lean towards zk-STARKs or other transparent proofs.

6.4 Future Work

There are multiple promising avenues to extend and enhance the work presented in this thesis:

- **Stronger Cryptographic Guarantees:**

An important direction is to eliminate the need for any trust in setup or cryptographic assumptions that might be contentious in the long term. Zeeperio currently uses KZG polynomial commitments (and thus a trusted setup and the elliptic curve pairing assumptions). In the future, one could replace this with a transparent proof system that requires no trusted setup and offers post-quantum security such as zk-STARKs, or alternative polynomial commitment schemes. While those approaches today have higher overhead, ongoing research is rapidly improving their efficiency. Adapting Zeeperio's constraints to a STARK or another transparent SNARK would remove the reliance on the SRS and make the protocol more robust against advances in cryptography (quantum attacks). The modular design of our protocol means it can be generalized to support different commitment schemes with relatively few changes. Exploring these options would enhance the trustworthiness and longevity of the system.

- **Full E2E Verifiability:**

Another important line of work is to incorporate stronger ballot privacy and coercion resistance into the protocol. As noted, Zeeperio does not yet guarantee voter privacy against a fully malicious authority. It assumes the authority behaves at least semi-honestly with respect to keeping ballots secret. Future enhancements could integrate cryptographic privacy mechanisms so that even the election authority cannot link voters to votes or learn more than the tally. One approach could be to combine Zeeperio with a trusted execution environment (TEE) or secure multiparty computation to

handle ballot encryption/decryption in a way that the proof can attest to correctness without revealing votes. For example, a TEE could be used to randomize and shuffle ballots before proof generation, thereby achieving a stronger secrecy guarantee while still outputting a proof that the tally is correct. Alternatively, one could extend the zero-knowledge property to prove cast-as-intended properties (that each encrypted ballot corresponds to a valid voting action by the voter) and voter anonymity properties, moving Zeeperio toward a fully E2E-verifiable voting system. Achieving these properties would fulfill the ultimate goal of verifiability with no caveats, where both integrity and privacy are enforced cryptographically. This is a challenging task, as added privacy constraints often significantly increase proof complexity, but it is a vital step for real-world adoption in high-stakes elections.

- **Generality and Beyond Voting:**

The techniques developed in Zeeperio may be applied to domains outside of the specific voting protocol we considered. One future work direction is to generalize the Zeeperio approach to other types of elections or even other verification problems. Within the realm of voting, this could mean adapting the proof constraints to different voting schemes (ranked-choice voting, or protocols that involve mix-nets) so that they too can benefit from automated verification. Beyond voting, any process that produces an outcome based on secret inputs and requires public auditability could, in principle, use a similar approach. This involves generating a succinct proof of correctness that a smart contract or public verifier can check. Examples might include software supply-chain audits, digital lottery drawings, or decentralized governance decisions. By changing the underlying arithmetic constraints, one could build a library of verifiable *proof-of-computation* gadgets for various processes. Zeeperio's success in the voting context is an encouraging proof-of-concept for this broader applicability.

- **Scalability and Performance Optimization:**

While our implementation handled 100k ballots with reasonable resources, future work should continue to optimize performance. This includes improving the prover efficiency and exploring optimizations in the constraint system. Reducing proving time and memory usage would make it easier for election authorities to generate proofs quickly after polls close. Additionally, as blockchain technology evolves, one could integrate Zeeperio with Layer-2 scaling solutions or specialized verification circuits to further reduce on-chain costs. Investigating the trade-offs between proof size, proving time, and verification cost could lead to an even more balanced and practical system for different election sizes and platforms.

In conclusion, this thesis has demonstrated a viable path toward autonomous election verification using cryptographic proofs. Zeeperio shows that it is possible to bridge the gap between cryptographic voting protocols and real-world election practice by introducing automation and efficiency in verification. By addressing the future work outlined above, we can move even closer to the ideal of elections that are fully transparent and trustworthy. Such advancements will help ensure that as voting systems modernize, they do so in a way that upholds voter trust and democratic legitimacy through verification. The hope is that the contributions of this work serve as a stepping stone toward that future of truly verifiable elections.

References

- [1] A. Essex, J. Clark, U. Hengartner, and C. Adams, “Eperio: Mitigating technical complexity in cryptographic election verification,” Cryptology ePrint Archive, Paper 2012/178, 2012. [Online]. Available: <https://eprint.iacr.org/2012/178>
- [2] J. Benaloh, “End-to-end verifiability,” *Annual Review of Cybersecurity*, vol. 1, pp. 101–117, 2014.
- [3] R. Carback, D. Chaum, J. Clark, J. Conway, A. Essex, P. S. Herrnson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen *et al.*, “Scantegrity {II} municipal election at takoma park: The first {E2E} binding governmental election with ballot privacy,” in *19th USENIX Security Symposium (USENIX Security 10)*, 2010.
- [4] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman, “Scantegrity ii: end-to-end verifiability for optical scan election systems using invisible ink confirmation codes,” in *Proceedings of the Conference on Electronic Voting Technology*, ser. EVT’08. USA: USENIX Association, 2008.
- [5] J. Groth, “On the size of pairing-based non-interactive arguments,” in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2016, pp. 305–326.
- [6] M. Harrison and T. Haines, “On the applicability of starks to counted-as-collected verification in existing homomorphic e-voting systems,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2024, pp. 50–65.
- [7] D. Chaum, “Secret-ballot receipts: True voter-verifiable elections,” *IEEE security & privacy*, vol. 2, no. 1, pp. 38–47, 2004.
- [8] S. Kremer, M. Ryan, and B. Smyth, “Election verifiability in electronic voting proto-

- cols,” in *European Symposium on Research in Computer Security*. Springer, 2010, pp. 389–404.
- [9] J. D. C. Benaloh, *Verifiable secret-ballot elections*. Yale University, 1987.
- [10] S. T. Ali and J. Murray, “An overview of end-to-end verifiable voting systems,” *CoRR*, vol. abs/1605.08554, 2016. [Online]. Available: <http://arxiv.org/abs/1605.08554>
- [11] A. Regenscheid, “End-to-end (e2e) verifiable protocols for voting systems,” https://www.eac.gov/sites/default/files/2023-01/e2e-draft-for-tgdc_508.pdf, 2023, presentation to the TGDC, Jan. 26, 2023.
- [12] P. B. Rønne, P. Y. A. Ryan, and B. Smyth, “Cast-as-intended: A formal definition and case studies,” in *Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers*. Berlin, Heidelberg: Springer-Verlag, 2021, p. 251–262. [Online]. Available: https://doi.org/10.1007/978-3-662-63958-0_22
- [13] S. Guasch and P. Morillo, “How to challenge and cast your e-vote,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 130–145.
- [14] C. Culnane, P. Y. A. Ryan, S. Schneider, and V. Teague, “vvote: a verifiable voting system,” 2015. [Online]. Available: <https://arxiv.org/abs/1404.6822>
- [15] M. Hirt and K. Sako, “Efficient receipt-free voting based on homomorphic encryption,” in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000, pp. 539–556.
- [16] J. Benaloh, “Simple verifiable elections,” in *Proceedings of the 2006 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT’06)*. Vancouver, BC, Canada: USENIX Association, Aug. 2006. [Online]. Available: https://www.usenix.org/legacy/event/evt06/tech/full_papers/benaloh/benaloh.pdf

- [17] J. Benaloh and D. Tuinstra, “Receipt-free secretballot elections (extended abstract).” *stoc 94: Proceedings of the twenty-sixth annual acm symposium on theory of computing,* 1994.
- [18] R. L. Rivest and W. D. Smith, “Three voting protocols: Threeballot, vav, and twin,” *USENIX/ACCURATE Electronic Voting Technology (EVT 2007)*, 2007.
- [19] T. Haines and B. Smyth, “Surveying definitions of coercion resistance,” *Cryptology ePrint Archive*, 2019.
- [20] W. Lueks, I. Querejeta-Azurmendi, and C. Troncoso, “{VoteAgain}: A scalable coercion-resistant voting system,” in *29th USENIX security symposium (USENIX Security 20)*, 2020, pp. 1553–1570.
- [21] D. Basin, J. Dreier, S. Giampietro, and S. Radomirović, “Verifying table-based elections,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2632–2652. [Online]. Available: <https://doi.org/10.1145/3460120.3484555>
- [22] D. Chaum, R. T. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, A. T. Sherman, and P. L. Vora, “Scantegrity ii: end-to-end verifiability by voters of optical scan elections through confirmation codes,” *Trans. Info. For. Sec.*, vol. 4, no. 4, p. 611–627, Dec. 2009. [Online]. Available: <https://doi.org/10.1109/TIFS.2009.2034919>
- [23] E. Shen, “End-to-end verifiability for optical scan voting systems,” Ph.D. dissertation, Massachusetts Institute of Technology, 2008.
- [24] C. Kempka, “Matters of coercion-resistance in cryptographic voting schemes,” Ph.D. dissertation, 2014.
- [25] R. Kuesters, T. Truderung, and A. Vogt, “Proving coercion-resistance of scantegrity

- II,” Cryptology ePrint Archive, Paper 2010/502, 2010. [Online]. Available: <https://eprint.iacr.org/2010/502>
- [26] C. Z. Acemyan, P. Kortum, M. D. Byrne, and D. S. Wallach, “Usability of voter verifiable, end-to-end voting systems: Baseline data for helios, {Prêt}{à} voter, and scantegrity {II},” in *2014 electronic voting technology workshop/workshop on trustworthy elections (EVT/WOTE 14)*, 2014.
- [27] M. R. Clarkson, S. Chong, and A. C. Myers, “Civitas: Toward a secure voting system,” in *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE, 2008, pp. 354–368.
- [28] V. Iovino, A. Rial, P. B. Rønne, and P. Y. Ryan, “Using selene to verify your vote in jcyj,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 385–403.
- [29] E. McMurtry, O. Pereira, and V. Teague, “When is a test not a proof?” in *European Symposium on Research in Computer Security*. Springer, 2020, pp. 23–41.
- [30] S. T. Ali and J. Murray, “An overview of end-to-end verifiable voting systems,” *Real-world electronic voting*, pp. 189–234, 2016.
- [31] B. Adida, “Helios: Web-based open-audit voting.” in *USENIX security symposium*, vol. 17, 2008, pp. 335–348.
- [32] B. Smyth and V. Cortier, “D4-1. formal description of our case study: Helios 2.0.”
- [33] O. Pereira, “Internet voting with helios,” in *Real-World Electronic Voting*. Auerbach Publications, 2016, pp. 293–324.
- [34] S. Neumann, M. M. Olembo, K. Renaud, and M. Volkamer, “Helios verification: To alleviate, or to nominate: Is that the question, or shall we have both?” in

International Conference on Electronic Government and the Information Systems Perspective. Springer, 2014, pp. 246–260.

- [35] F. Karayumak, M. M. Olembo, M. Kauer, and M. Volkamer, “Usability analysis of helios—an open source verifiable remote electronic voting system,” in *2011 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 11)*, 2011.
- [36] S. Baloglu, “Formal verification of verifiability in e-voting protocols,” 2023.
- [37] D. Y. M. Del Blanco and M. Gascó, “A protocolized, comparative study of helios voting and scytl/ivote,” in *2019 Sixth International Conference on eDemocracy & eGovernment (ICEDEG)*. IEEE, 2019, pp. 31–38.
- [38] V. Cortier and B. Smyth, “Attacking and fixing helios: An analysis of ballot secrecy,” *Journal of Computer Security*, vol. 21, no. 1, pp. 89–148, 2013.
- [39] B. Adida, O. De Marneffe, O. Pereira, J.-J. Quisquater *et al.*, “Electing a university president using open-audit voting: Analysis of real-world use of helios,” *EVT/WOTE*, vol. 9, no. 10, p. 10, 2009.
- [40] M. Backes, C. Hammer, D. Pfaff, and M. Skoruppa, “Implementation-level analysis of the javascript helios voting client,” in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 2071–2078.
- [41] L. P. Alonso, M. Gasco, D. Y. M. Del Blanco, J. Á. H. Alonso, J. Barrat, and H. A. Moreton, “E-voting system evaluation based on the council of europe recommendations: Helios voting,” *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 1, pp. 161–173, 2018.
- [42] A. Kiayias, T. Zacharias, and B. Zhang, “Auditing for privacy in threshold pke e-voting,” *Information & Computer Security*, vol. 25, no. 1, pp. 100–116, 2017.

- [43] S. Bell, J. Benaloh, M. D. Byrne, D. DeBeauvoir, B. Eakin, P. Kortum, N. McBurnett, O. Pereira, P. B. Stark, D. S. Wallach *et al.*, “{STAR-Vote}: A secure, transparent, auditable, and reliable voting system,” in *2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 13)*, 2013.
- [44] A. W. Appel and P. B. Stark, “Evidence-based elections: Create a meaningful paper trial, then audit,” *Geo. L. Tech. Rev.*, vol. 4, p. 523, 2019.
- [45] T. Kraavi and J. Willemson, “Proving vote correctness in the ivxv internet voting system,” *Scientific Reports*, vol. 15, no. 1, p. 31793, 2025.
- [46] M. Lindeman and P. B. Stark, “A gentle introduction to risk-limiting audits,” *IEEE Security & Privacy*, vol. 10, no. 5, pp. 42–49, 2012.
- [47] P. B. Stark, “An introduction to risk-limiting audits and evidence-based elections,” *Testimony prepared for the California Little Hoover Commission. July*, vol. 2, 2018.
- [48] Q. Yang, “Coercion-resistance in electronic voting: design and analysis,” Ph.D. dissertation, PhD Thesis. France: Universit e de Lorraine, 2023.
- [49] R. L. Rivest and M. Virza, “Software independence revisited,” in *Real-World Electronic Voting*. Auerbach Publications, 2016, pp. 19–34.
- [50] W. Quesenbery, S. Chapman, C. Patten, R. Sprenggiaro, and S. Laskowski, “Voter review and verification of ballots,” 2023.
- [51] A. Appel, R. DeMillo, and P. Stark, “Ballot-marking devices (bmds) cannot assure the will of the voters. april 21, 2019,” *Available at SSRN 3375755*.
- [52] J. Benaloh, M. Naehrig, O. Pereira, and D. S. Wallach, “{ElectionGuard}: a cryptographic toolkit to enable verifiable elections,” in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5485–5502.

- [53] A. Zanga, “„using electionguard for secure remote voting on untrusted devices “,” 2020.
- [54] “Electionguard official specifications,” <https://electionguard.vote/spec/>, ElectionGuard, 2025.
- [55] B. Smith, “Another step in testing electionguard,” <https://blogs.microsoft.com/on-the-issues/2020/02/17/wisconsin-electionguard-polls/>, Microsoft, 2020.
- [56] T. Burt, “Protecting democratic elections through secure, verifiable voting,” <https://blogs.microsoft.com/on-the-issues/2019/05/06/protecting-democratic-elections-through-secure-verifiable-voting/>, 2019, microsoft On the Issues.
- [57] Voatz, Inc., “Voatz mobile voting platform: An overview – security, identity, auditability,” <https://voatz.com/wp-content/uploads/2020/07/voatz-security-whitepaper.pdf>, Voatz, Inc., Tech. Rep. Version 1.1, 2020.
- [58] M. A. Specter, J. Koppel, and D. Weitzner, “The ballot is busted before the blockchain: A security analysis of voatz, the first internet voting application used in {US}. federal elections,” in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1535–1553.
- [59] D. Guido, “Our full report on the voatz mobile voting platform,” *Trail of Bits*, 2020.
- [60] S. Edwards, J. Smith, D. Guido, and E. Sultanik, “Voatz security assessment, volume i: Technical findings,” <https://verifiedvoting.org/wp-content/uploads/2020/10/voatz-securityreview.pdf>, Trail of Bits, Tech. Rep., 2020.
- [61] B. Goldstein, “Audit finds severe vulnerabilities in voatz mobile voting app,” <https://statescoop.com/audit-finds-severe-vulnerabilities-voatz-mobile-voting-app/>, 2020.
- [62] M. Maciag, “Detailed audit of voatz’ voting app confirms security flaws,” <https://>

www.govtech.com/biz/detailed-audit-of-voatz-voting-app-confirms-security-flaws.html, 2020, accessed: 2025-12-20.

- [63] Voatz, Inc., “Audits,” <https://new.voatz.com/audits/>, 2025.
- [64] M. Korolov, “Voatz, mit researchers spar over blockchain e-voting app,” <https://www.techtargget.com/searchsecurity/news/252478923/Voatz-MIT-researchers-spar-over-blockchain-e-voting-app>, 2020.
- [65] L. Moore and N. Sawhney, “Under the hood: The west virginia mobile voting pilot,” <https://www.nass.org/sites/default/files/2019-02/white-paper-voatz-nass-winter19.pdf>, Voatz, Inc., Tech. Rep., 2019.
- [66] B. Rathbun, “The rarity of realpolitik: What bismarck’s rationality reveals about international politics,” *International Security*, vol. 43, no. 1, pp. 7–55, 2018.
- [67] D. of Justice Canada, “Section 3 – democratic rights,” <https://www.justice.gc.ca/eng/csjsjc/rfc-dlc/ccrf-ccd1/check/art3.html>, 1999.
- [68] L. of Parliament, “Electoral rights: Charter of rights and freedoms (90-5e),” <https://publications.gc.ca/Collection-R/LoPBdP/CIR/905-e.htm>, 2001.
- [69] “Harper v. canada (attorney general),” <https://decisions.scc-csc.ca/scc-csc/scc-csc/en/item/2146/index.do>, 2004.
- [70] Centre for Constitutional Studies, “Harper v canada (2004) – third party election advertising limits in federal election campaigns,” 2012.
- [71] “Opitz v. wrzesnewskyj,” <https://decisions.scc-csc.ca/scc-csc/scc-csc/en/item/12635/index.do>, 2012.
- [72] International Foundation for Electoral Systems, “Opitz v. wrzesnewskyj – election judgment summary,” <https://electionjudgments.org/en/entity/157vugn16kl/references>, 2012.

- [73] “Municipal elections act, 1996, s.o. 1996, c. 32, sch.” <https://www.ontario.ca/laws/statute/96m32>, 1996.
- [74] T. of Clearview, “2026 municipal & school board election voting method (report ls-023-2024),” <https://www.clearview.ca/file/ls-023-2024-2026-municipal-school-board-election-voting-methodpdf>, 2024.
- [75] A. Essex *et al.*, “Online voting in ontario municipalities: A standards-based security analysis of vendor systems,” <https://pub-guelph.escribemeetings.com/filestream.ashx?DocumentId=47779>, Whisper Lab / Western University, Tech. Rep., 2024.
- [76] CBC News, “Security flaws left ontario’s 2022 municipal online elections highly vulnerable, study finds,” 2025.
- [77] A. Essex, “Protect municipal democracy by strengthening online voting security,” *Policy Options*, 2022.
- [78] E. Canada, “Electoral integrity framework – principles and objectives,” <https://www.elections.ca/content.aspx?section=abo&dir=fra&document=index&lang=e>, 2025.
- [79] A. Cardillo, N. Akinyokun, and A. Essex, “Online voting in ontario municipal elections: a conflict of legal principles and technology?” in *International Joint Conference on Electronic Voting*. Springer, 2019, pp. 67–82.
- [80] A. Essex, “Internet voting in canada: A cyber security perspective,” <https://www.ourcommons.ca/content/Committee/421/ERRE/Brief/BR8610535/br-external/EssexAleksander-e.pdf>, House of Commons Special Committee on Electoral Reform, Tech. Rep., 2016.
- [81] P. B. Stark and D. Wagner, “Evidence-based elections,” *IEEE Security & Privacy*, vol. 10, no. 5, pp. 33–41, 2012.
- [82] R. L. Rivest, “On the notion of ‘software independence’ in voting systems,”

Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 366, no. 1881, pp. 3759–3767, 08 2008. [Online]. Available: <https://doi.org/10.1098/rsta.2008.0149>

- [83] P. of Canada, “The canadian electoral system: Questions and answers,” https://lop.parl.ca/sites/PublicWebsite/default/en_CA/ResearchPublications/202302E, 2023, section 2.1.
- [84] E. Canada, “Electoral integrity framework: Principles and objectives,” <https://www.elections.ca/content.aspx?section=abo&dir=fra%2Fcons&document=index&lang=e>, 2025.
- [85] E. Ontario, “Modernizing ontario’s electoral process: 2018 general election – post-event report,” <https://www.elections.on.ca/content/dam/NGW/sitecontent/2019/Reports/2018%20General%20Election%20-%20Post-Event%20Report.pdf>, 2019.
- [86] —, “Recommendations for management standards on electronic poll books and vote tabulators,” <https://www.elections.on.ca/content/dam/NGW/sitecontent/advisory-committee/reports/Recommendations%20for%20Management%20Standards%20on%20Electronic%20Poll%20Books%20and%20Vote%20Tabulators.pdf>, 2023.
- [87] P. B. Appel, Andrew W. Stark, “Evidence-based elections: Create a meaningful paper trail, then audit digital technologies & voting,” p. 523, 2019-2020.
- [88] L. Garland, M. Lindeman, N. McBurnett, J. Morrell, M. Schneider, and S. Singer, “Principles and best practices for post-election tabulation audits,” 2018.
- [89] A. Katawazy, “Identifying challenges and advantages of internet voting and its impact on voter turnout in ontario municipal elections,” Master’s thesis, Western University, 2022.
- [90] S. Dzieduszycka-Suinat, J. R. Murray, J. Kiniry, D. Zimmerman, D. Wagner,

- P. Robinson, A. Foltzer, and S. Morina, “The future of voting: End-to-end verifiable internet voting - specification and feasibility study,” U.S. Vote Foundation, Tech. Rep., 2015. [Online]. Available: https://usvotefoundation-drupal.s3.amazonaws.com/prod/E2EVIV_full_report.pdf
- [91] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed. John Wiley & Sons, 1996.
- [92] Dominion Voting Systems, “Dominion voting statement regarding internet voting service slowdown affecting ontario municipalities,” <https://www.penetanguishene.ca/media/y4iob2ah/statement-dominion-voting-ontario-iv-issue-221018-002.pdf>, October 2018.
- [93] CBC News, “Online voting is growing in canada, raising calls for clear standards,” October 2022.
- [94] Global News, “Online voting in 51 ontario municipalities marred by election-day issues,” October 2018.
- [95] A. Essex, “Now is the time to regulate voting technologies in canada,” *The Globe and Mail*, August 2021.
- [96] S. Chen, R. Wang, X. Wang, and K. Zhang, “Side-channel leaks in web applications: A reality today, a challenge tomorrow,” 01 2010, pp. 191–206.
- [97] A. Belousova, F. Marchiori, and M. Conti, “Inference attacks on encrypted online voting via traffic analysis,” 2025. [Online]. Available: <https://arxiv.org/abs/2509.15694>
- [98] CBC News, “Electronic voting blamed for quebec municipal election ‘disaster’,” 2006.
- [99] Office of the Information and Privacy Commissioner of Newfoundland and

- Labrador, “Internet voting – privacy and security risks,” <https://www.oipc.nl.ca/files/InternetVoting%E2%80%93PrivacyAndSecurityRisks.pdf>, Tech. Rep., 2020.
- [100] Council of Europe, “Guidelines on transparency of e-enabled elections,” <https://rm.coe.int/CoERMPublicCommonSearchServices/DisplayDCTMContent?documentId=090000168059bdf6>, 2011.
- [101] —, “Guidelines on the implementation of the provisions of recommendation cm/rec(2017)5 on standards for e-voting,” <https://rm.coe.int/0900001680703319>, 2017.
- [102] Digital Governance Standards Institute, “Online voting – part 1: Implementation of online voting in canadian municipal elections (notice of intent),” <https://scc-ccn.ca/standards/notices-of-intent/digital-governance-standards-institute/online-voting-part-1>, 2024.
- [103] —, “Can/dgsi 111-1: Online voting – part 1: Implementation of online voting in canadian municipal elections,” <https://dgc-cgn.org/product/can-dgsi-111-1/>, 2024.
- [104] W. Jamroga *et al.*, “A declaration of software independence,” *SWiND workshop*, 2021.
- [105] G. Bernhard *et al.*, “Receipt-free electronic voting from zk-snarks,” in *Proceedings of ESORICS*, 2017.
- [106] A. as listed, “Dispute-free scalable open vote network using zk-snarks,” *Cryptology ePrint Archive / conference proceedings*, 2022.
- [107] L.-H. Merino, S. Colombo, R. Reyes, A. Azhir, S. Mishra, P. Tennage, M. A. Raeisi, H. Zhang, J. R. Allen, B. Tellenbach *et al.*, “Trip: Coercion-resistant registration for e-voting with verifiability and usability in voteqral,” in *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, 2025, pp. 834–874.

- [108] G. of Canada, “Online voting: A path forward for federal elections,” <https://www.canada.ca/en/democratic-institutions/services/reports/online-voting-path-forward-federal-elections.html>, Privy Council Office, Tech. Rep., 2017.
- [109] R. Lidl and H. Niederreiter, *Finite Fields*, 2nd ed., ser. Encyclopedia of Mathematics and its Applications. Cambridge: Cambridge University Press, 1997, vol. 20.
- [110] S. Lang, *Algebra*, 3rd ed., ser. Graduate Texts in Mathematics. New York: Springer, 2002, vol. 211.
- [111] G. L. Mullen and D. Panario, “Polynomials over finite fields,” in *Handbook of Finite Fields*, G. L. Mullen and D. Panario, Eds. Chapman and Hall/CRC, 2013, ch. 3.
- [112] J.-L. Lagrange, *Théorie des fonctions analytiques*. Paris: Imprimerie de la République, 1797.
- [113] J. T. Schwartz, “Fast probabilistic algorithms for verification of polynomial identities,” *Journal of the ACM (JACM)*, vol. 27, no. 4, pp. 701–717, 1980.
- [114] D. S. Dummit and R. M. Foote, *Abstract Algebra*, 3rd ed. John Wiley & Sons, 2004.
- [115] J. H. Silverman, *The Arithmetic of Elliptic Curves*, 2nd ed., ser. Graduate Texts in Mathematics. Springer, 2009, vol. 106.
- [116] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 3rd ed. Boca Raton: CRC Press, 2020.
- [117] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2004.
- [118] D. Boneh and X. Boyen, “Short signatures without random oracles,” in *Advances*

- in Cryptology – EUROCRYPT 2004*, ser. Lecture Notes in Computer Science, vol. 3027. Springer, 2004, pp. 56–73.
- [119] S. D. Galbraith, K. G. Paterson, and N. P. Smart, “Pairings for cryptographers,” *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113–3121, 2008.
- [120] G. Fuchsbauer, “The algebraic group model and its applications,” in *Advances in Cryptology – CRYPTO 2018*, ser. Lecture Notes in Computer Science, vol. 10991. Springer, 2018, pp. 33–62.
- [121] T. P. Pedersen, “Non-interactive and information-theoretic secure verifiable secret sharing,” in *Advances in Cryptology – CRYPTO ’91*, ser. Lecture Notes in Computer Science, vol. 576. Springer, 1992, pp. 129–140.
- [122] A. Kate, G. M. Zaverucha, and I. Goldberg, “Constant-size commitments to polynomials and their applications,” in *Advances in Cryptology – ASIACRYPT 2010*, ser. Lecture Notes in Computer Science, vol. 6477. Springer, 2010, pp. 177–194.
- [123] O. Goldreich, *Foundations of Cryptography, Volume 1: Basic Tools*. Cambridge University Press, 2001.
- [124] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, “PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge,” *Cryptology ePrint Archive*, Paper 2019/953, 2019. [Online]. Available: <https://eprint.iacr.org/2019/953>
- [125] R. J. Fateman, “The fast fourier transform,” University of California, Berkeley, Lecture Notes CS 282, 1997, explains radix-2 FFT algorithm requiring domain size to be power of two for efficiency. [Online]. Available: <https://people.eecs.berkeley.edu/~fateman/lectures/fft.ps>
- [126] E. Ben-Sasson, A. Chiesa, and N. Spooner, “Interactive oracle proofs,” in *Theory*

of Cryptography, M. Hirt and A. Smith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 31–60.

- [127] J. Liang, D. Hu, P. Wu, Y. Yang, Q. Shen, and Z. Wu, “Sok: Understanding zk-snarks: The gap between research and practice,” in *Proceedings of the 34th USENIX Security Symposium*. Seattle, WA, USA: USENIX Association, 2025. [Online]. Available: <https://www.usenix.org/system/files/usenixsecurity25-liang-sok.pdf>
- [128] A. Fiat and A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” in *Advances in Cryptology—CRYPTO’ 86*, A. M. Odlyzko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194.
- [129] Etherscan, “Ethereum node tracker,” <https://etherscan.io/nodetracker>, 2025, accessed: 2025-11-10.
- [130] Ethereum Foundation. (2025) Ethereum proof-of-stake attack and defense. [Online]. Available: <https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/attack-and-defense/>
- [131] I. E. T. F. (IETF), “Pairing-friendly curves,” Accessed: Oct. 9, 2025, IETF, Tech. Rep. draft-irtf-cfrg-pairing-friendly-curves-07, 2020, section 3.2, BN254 curve parameters. [Online]. Available: <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-pairing-friendly-curves-07>
- [132] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, “Sonic: Zero-knowledge snarks from linear-size universal structured reference strings,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2019, pp. 2111–2128.
- [133] F. Wang, S. Cohneney, and J. Bonneau, “SoK: Trusted setups for powers-of-tau strings,” *Cryptology ePrint Archive*, Paper 2025/064, 2025. [Online]. Available: <https://eprint.iacr.org/2025/064>

- [134] E. Canada, “Federal electoral district fact sheets,” 2025, accessed: 2025-09-18. [Online]. Available: <https://electionsanddemocracy.ca/your-classroom/resources/geography-elections/district-fact-sheets>
- [135] B. de la Calle Ysern and P. Galán del Sastre, “A lagrange interpolation with preprocessing to nearly eliminate oscillations,” *Numerical Algorithms*, vol. 97, no. 4, pp. 2051–2082, 2024. [Online]. Available: <https://doi.org/10.1007/s11075-024-01778-z>
- [136] L. Euler, “Theorematum quorundam ad numeros primos spectantium demonstratio,” *Proceedings of the St. Petersburg Academy*, 1736.
- [137] C. Reitwiessner, “Eip-196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128,” Ethereum Foundation, Ethereum Improvement Proposal 196, 2017. [Online]. Available: <https://eips.ethereum.org/EIPS/eip-196>
- [138] V. Buterin, “Eip-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128,” Ethereum Foundation, Ethereum Improvement Proposal 197, 2017. [Online]. Available: <https://eips.ethereum.org/EIPS/eip-197>
- [139] “Ethereum contract abi specification,” Ethereum Foundation, Specification, 2017. [Online]. Available: <https://docs.soliditylang.org/en/v0.8.25/abi-spec.html>
- [140] W. Labs, “Optimizing gas in solidity smart contracts: Choosing the right storage,” <https://blog.web3labs.com/optimizing-gas-in-solidity-smart-contracts-choosing-the-right-storage/>, 2024.
- [141] “Foundry—ethereum development framework,” <https://getfoundry.sh>.
- [142] GetBlock, “What is sepolia? a beginner’s guide to ethereum test networks,” <https://getblock.medium.com/>

[what-is-sepolia-a-beginners-guide-to-ethereum-test-networks-866663a26698](https://www.researchgate.net/publication/3866663a26698),
2024.

- [143] N. Huber, R. Kuesters, J. Liedtke, and D. Rausch, “ZK-SNARKs for ballot validity: A feasibility study,” Cryptology ePrint Archive, Paper 2024/1902, 2024. [Online]. Available: <https://eprint.iacr.org/2024/1902>
- [144] Reilabs, “Implementing trusted setup ceremony for ethereum’s eip-4844,” <https://reilabs.io/blog/implementing-trusted-setup-ceremony-for-ethereums-eip-4844/>, Sep. 2024.
- [145] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, “Recursive composition and bootstrapping for snarks and proof-carrying data,” in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, 2013, pp. 111–120.

Curriculum Vitae

Name: Aikamdeep Malhotra

Post-Secondary Education and Degrees: University of Western Ontario
London, ON, Canada
2019 - 2023 BESc

University of Western Ontario
London, ON, Canada
2024 - 2025 M.E.Sc

Related Work Experience: Teaching Assistant
The University of Western Ontario
2024 - 2025

Research Assistant
The University of Western Ontario
2024 - 2025

Publications:

Aikamdeep Malhotra, Aleksander Essex, and Jeremy Clark, Zeperio: Verifying Governmental Elections with Ethereum, Financial Cryptography and Data Security (FC). Workshop on Trusted Smart Contracts (WTSC), 2026. Accepted.