University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Mixnets in a Distributed Ledger Remote Electronic Voting System

Moritz Eck
Switzerland
Student ID: 14-715-296

Supervisor: Christian Killer, Bruno Rodrigues
Date of Submission: March 22, 2021

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

The integrity and fair execution of privacy-preserving votes and elections is a cornerstone of modern democracy. Changes to the voting process are delicate and highly debated, including the introduction of remote electronic voting (REV). REV poses a unique set of challenges as it allows citizens to cast their ballots from an uncontrolled environment (*e.g.*, mobile phone). The legal and technical requirements increase the difficulty in digitizing votes and elections as the preservation of privacy required by law is in direct contrast to the verifiability of the system. Extensive research has been conducted over the past years trying to satisfy these requirements by applying various cryptographic techniques. Although many REV systems have been proposed in literature, few have been implemented practically. Additionally, almost all of them pursue a centralized approach, in which a single authority is trusted to handle the votes according to protocol. By distributing the trust among the stakeholders in the system, similar to the federal structure of Switzerland, and thereby preventing a single point of failure, this work takes a different approach. A key component in the success is a distributed ledger utilized as public bulletin board to guarantee transparency, integrity and robustness through decentralization. Using established cryptographic techniques (*e.g.*, mixnets, zero-knowledge proofs) this work further enhances the scope of Provotum, a secure and verifiable distributed REV system developed at CSG@IfI, by enabling elections and shifting the computational burden from the voter to the voting infrastructure, allowing participation from any device. The proposed voting protocol is implemented in a proof-of-concept and evaluated in terms of privacy, verifiability and scalability. The results show that the system can scale to nationwide elections and votes, as the computational complexity scales linearly with the number of ballots cast. Finally, possibilities for improvement and expansion are identified.

Kurzfassung

Die Integrität und faire Durchführung von Abstimmungen und Wahlen unter Wahrung der Privatsphäre ist ein Eckpfeiler der modernen Demokratie. Änderungen am Wahlverfahren sind heikel und werden heftig diskutiert, darunter auch die Einführung der Internet-gestützten, elektronischen Stimmabgabe (REV). REV stellt eine einzigartige Reihe von Herausforderungen dar, da es den Bürgern erlaubt, ihre Stimme in einer unkontrollierten Umgebung (*z.B.* von einem Mobiltelefon) abzugeben. Die rechtlichen und technischen Anforderungen erhöhen die Schwierigkeit bei der Digitalisierung von Abstimmungen und Wahlen, da das gesetzlich geforderte Stimmgeheimnis und die Privatsphäre des Wählers in direktem Gegensatz zur Verifizierbarkeit des Systems steht. In den letzten Jahren wurden umfangreiche Forschungsarbeiten durchgeführt, die versuchen, diese Anforderungen durch die Anwendung verschiedener kryptographischer Techniken zu erfüllen. Obwohl in der Literatur viele REV-Systeme vorgeschlagen wurden, sind nur wenige davon praktisch umgesetzt worden. Ausserdem verfolgen fast alle einen zentralisierten Ansatz, bei dem einer einzigen Instanz vertraut wird, die Stimmen protokollgemäss zu behandeln. Diese Arbeit verfolgt einen anderen Ansatz, indem sie das Vertrauen auf die Beteiligten im System verteilt, ähnlich der föderalen Struktur der Schweiz, und dadurch einen Single Point of Failure verhindert. Eine Schlüsselkomponente für den Erfolg ist das öffentliche elektronische Anschlagsbrett, welches mittels Distributed-Ledger Technologie umgesetzt wird, um Transparenz, Integrität und Robustheit durch Dezentralisierung zu gewährleisten. Unter Verwendung etablierter kryptographischer Techniken (*z.B.* Mixnets, Zero-Knowledge Proofs) erweitert diese Arbeit den Anwendungsbereich von Provotum, einem sicheren und überprüfbaren verteilten REV-System, das am CSG@IfI entwickelt wurde, indem es Wahlen ermöglicht und die Rechenlast vom Wähler auf die Wahlinfrastruktur verlagert, was die Teilnahme von jedem Gerät aus ermöglicht. Als Machbarkeitsnachweis wird das vorgeschlagene Wahlprotokoll in einem Prototyp implementiert und in Bezug auf Privatsphäre, Verifizierbarkeit und Skalierbarkeit evaluiert. Die Ergebnisse zeigen, dass das System auf landesweite Wahlen und Abstimmungen skalieren kann, da die Rechenkomplexität linear mit der Anzahl der abgegebenen Stimmen skaliert. Abschliessend werden Möglichkeiten zur Verbesserung und Erweiterung aufgezeigt.

Acknowledgments

I would like to express my gratefulness to the people that have supported me over the years, in order for me to be able to accomplish this goal.

First and foremost, I would like to thank Christian Killer for guiding me in the right direction, his valuable feedback on ideas that I proposed, and for the enriching discussions we have had. Also, I want to thank Prof. Dr. Burkhard Stiller for the opportunities throughout my master studies and the possibility to realize this thesis under the supervision of the Communication Systems Group.

Moreover, I would like to thank my parents for their ongoing support and for always believing in me. And, Sina for her endless encouragement and making me smile.

I am very grateful for their support.

Contents

Abstract i

Kurzfassung iii

Acknowledgments v

1 Introduction 1

1.1 Project Goals and Contributions 2

1.2 Thesis Outline 2

2 Background 3

2.1 Voting Protocol Properties 3

2.1.1 Privacy 3

2.1.2 Verifiability 4

2.1.3 Practical Properties 5

2.1.4 Trust Assumptions & Trade-Offs 6

2.2 Distributed Ledger Technology 7

2.2.1 Permission Models & Consensus Mechanisms 7

2.2.2 DL as Public Bulletin Board 8

3 Cryptography **9**

3.1 Hash Functions . . . 9

3.2 Public Key Cryptography . . . 10

3.3 Homomorphic Encryption . . . 10

3.4 ElGamal Cryptosystem . . . 10

3.4.1 Security . . . 11

3.4.2 Key Generation, Encryption and Decryption . . . 11

3.4.3 Message Encoding . . . 12

3.5 Re-Encryption . . . 12

3.6 Mixnets . . . 13

3.6.1 Decryption Mixnets . . . 14

3.6.2 Re-Encryption Mixnets . . . 14

3.6.3 Summary . . . 15

3.7 Multi-Party Computation . . . 16

3.7.1 Distributed Key Generation . . . 16

3.7.2 Cooperative Decryption . . . 16

3.8 Zero-Knowledge Proofs . . . 18

3.8.1 Key Generation Proof . . . 19

3.8.2 Decryption Proof . . . 19

3.8.3 Re-Encryption Proof . . . 20

3.8.4 Shuffle Proof . . . 21

4 Related Work **27**

4.1 Application-level Privacy . . . 27

4.2 Network-level Privacy . . . 28

4.3 Everlasting Privacy . . . 29

4.4 Existing REV Systems . . . 30

4.5 Distributed Ledgers and REV . . . 31

5 System Design **33**

5.1 Stakeholders . . . 33

5.2 Voting Protocol . . . 35

5.2.1 Identity Management . . . 35

5.2.2 Pre-Voting Phase . . . 35

5.2.3 Voting Phase . . . 37

5.2.4 Post-Voting Phase . . . 38

6 Implementation **41**

6.1 Packages . . . 41

6.2 Documentation . . . 44

6.3 Technical Limitations . . . 44

7 Discussion and Evaluation **47**

7.1 Privacy . . . 47

7.1.1 Ballot Secrecy . . . 47

7.1.2 Receipt-Freeness . . . 48

7.1.3 Coercion-Resistance . . . 48

7.2 Verifiability . . . 49

7.2.1 Cast-as-Intended Verifiability . . . 49

7.2.2 Individual & Recorded-as-Cast Verifiability . . . 49

7.2.3 Counted-as-Recorded . . . 50

7.2.4 Summary . . . 50

7.3 Practical Properties . . . 50

7.3.1 Fairness . . . 51

7.3.2 Accountability . . . 51

7.3.3 Robustness . . . 51

7.3.4 Votes and Elections . . . 51

7.4 Scalability . . . 52

7.4.1 Theoretical Runtime Performance . . . 52

7.4.2 Benchmark Results . . . 53

8 Conclusion and Future Work **59**

8.1 Conclusion 59

8.2 Future Work 60

8.2.1 Cast-as-Intended (CaI) Verifiability 60

8.2.2 k/n Distributed Key Generation 60

8.2.3 Technical Improvements 60

8.2.4 Identity Management 61

Bibliography **62**

Abbreviations **73**

List of Figures **73**

List of Tables **76**

A Installation Guidelines **79**

B Contents of the CD **81**

Chapter 1

Introduction

The integrity and fair execution of privacy-preserving votes and elections is a cornerstone of modern democracy. Any changes to the voting process are delicate and highly debated, including the introduction of remote electronic voting (REV) [75]. The design and implementation of REV systems pose a unique set of challenges to be tackled and additional properties to be achieved. REV in particular allows citizens to cast their ballots remotely, from an uncontrolled device and in an uncontrolled environment (*e.g.*, from a web browser or mobile phone). The legal and technical requirements increase the difficulty in digitizing votes and elections as the preservation of privacy required by law is in direct contrast to the verifiability of the system. Another common issue with REV systems is that the existence of a public bulletin board (PBB) used to store, audit and trail every step of a vote is often formulated as an assumption in related work [64]. Due to these operational and practical issues, the Provotum project focuses on the design and architecture of such a PBB, including its distributed execution. Compared to other REV systems in which a centralized authority is trusted to handle the votes according to protocol (*i.e.*, no decrypting before the vote has ended, no publishing of voter specific information), the Provotum project takes a different approach. Trust is distributed among the stakeholders in the system, similar to the federal structure of Switzerland and thereby preventing single points of failure. A key component in this distribution of trust play distributed ledgers (DL), an exciting advancement in distributed systems research. DLs are an immutable, append-only, publicly-readable, distributed data structure making them an ideal candidate for PBBs in REV systems as the necessary transparency, robustness and integrity guarantee are inherent. In addition, a wide variety of approaches (*e.g.*, distributed key generation and zero-knowledge proofs) are explored towards the goal of achieving a fully decentralized REV system [66, 72, 73, 81, 82].

Research on verifiable REV systems can be grouped into three main categories; systems based on (1) homomorphic encryption (HE), (2) mixnets and (3) commitment schemes. REV systems utilizing HE make use of the mathematical properties of the underlying cryptosystem which allows the result of a vote to be computed without decrypting individual ballots. This approach has been thoroughly investigated so far in the Provotum project [72, 73, 81, 82]. The approach is conceptually simple but depending on the application, it does not scale past small numbers of participants. In addition, the solution is computationally expensive for the individual participants as a proof of validity must be

provided for each ballot cast. Alternatively, mixnets employ a different approach. They are based on cryptographically mixing the ballots to ensure unlinkability between the voter and her ballots. An advantage of using mixnets is that the computational burden is shifted from the voter to the voting infrastructure, allowing voters to participate from any device. Another important facet when compared to HE, is the decreased ballot complexity and that no ballot validity proof is necessary. For instance, elections and write-ins (for additional candidates) are easier to implement in mixnets because during the tallying the individual ballots are decrypted and counted, as opposed to just the final result (as in the case of HE). Still, mixnets have their own challenges, as proving that mixing was done honestly and correctly is computationally expensive. Improvements in this area are still an active field of research.

1.1 Project Goals and Contributions

The thesis main goal is to tackle the limitations of homomorphic tallying based REV systems (*e.g.*, Provotum 2.0 [72] and 3.0 [66]) by replacing them with a mixnet-based approach. To propose a more suitable system architecture, an analysis of prior work on verifiable mixnets is performed. The new system shall be designed using an extendable software architecture keeping in mind the desired privacy properties of REV systems. The prototype shall be developed with a focus on verifiability and scalability, evaluating if it can be applied for nationwide votes. Remaining limitations and other technical challenges must be documented and addressed such that they can be tackled in future work.

1.2 Thesis Outline

The remaining part of this thesis is structured as follows. Chapter 2 introduces the privacy, verifiability and practical properties required to asses REV systems. It establishes the most important concepts required to understand the remaining part of this thesis. Also, it provides a short introduction into distributed ledger technologies and how those can be leverage as a PBB. Next, Chapter 3 provides an in depth explanation of the cryptographic primitives applied in the proposed voting protocol. Chapter 4 presents prior academic and commercial work on REV systems with a focus on mixnets, its different variations and applications. Then, in Chapter 5, the system's architecture is denoted, the different stakeholders are introduced, as well as the voting protocol that this thesis is based on is explained. Chapter 6 describes the implementation and introduces the underlying components and packages that the system is composed of. Subsequently, in Chapter 7 the proposed system is evaluated in terms of privacy, verifiability, practicality as well as scalability. Finally, Chapter 8 concludes this thesis by summarizing its achievements and suggesting potential future research and implementation possibilities.

Chapter 2

Background

Any type of voting that involves electronic means can be considered as electronic voting [69]. The literature [103] differentiates between two major types that use an electronic apparatus to cast, record and count votes; electronic machine voting (EMV) in a fixed public place and electronic distance voting (EDV) from different locations. In the context of this thesis, we define the term remote electronic voting (REV) as a synonym of Internet Voting, a subtype of EDV in which voters cast their ballots in an uncontrolled environment. To be able to understand how REV systems work, knowledge of different concepts is required, which is introduced in this chapter. The first section, discusses the main properties central to REV systems and their inherent contradiction. The second part, introduces distributed ledger technology and how they are applicable in the context of REV systems.

2.1 Voting Protocol Properties

From the academic literature [67] and the legal framework in Switzerland [31, 29, 32], a set of desireable properties for REV systems has emerged over the past decades. The definition of these properties varies between authors. Therefore, this section aims to provide an overview of the definitions, necessary to discuss REV systems, as understood and interpreted in this work. Jonker [67] provides a broad overview of the various properties and forms the basis for the definitions. The properties can be group into two main categories, Privacy and Verifiability.

2.1.1 Privacy

Privacy in voting is considered a fundamental human right as stated in the Swiss Constitution [30] and the universal declaration of human rights [8]. In other words, keeping the content of a ballot secret is of utmost importance. Especially, considering that in a REV system the voters cast their votes in an uncontrolled environment and from an uncontrolled device (*e.g.*, mobile phone). In the literature [67], privacy is futher grouped into: (1) Ballot Secrecy (BP), (2) Receipt-Freeness (RP) and (3) Coercion-Resistance (CR).

Definition 1 *Ballot Secrecy. In a REV system with BS, it is impossible to determine for anyone (apart from the voter) how a voter has voted and to link cast ballots to a specific voter [19].*

Definition 2 *Everlasting Ballot Secrecy. Everlasting BS extends the notion of BS to a computationally unbounded adversary for whom it is also impossible to determine how a voter as voted and to link cast ballots to a specific voter [67].*

Mixnet- and HE-based REV systems are built upon the primary assumptions of asymmetric cryptographic, the intractability of mathematical problems, as explained in Sections 3.2 and 3.4.1. Should these assumptions no longer hold due to a computationally unbounded adversary appearing at some point in the future, BS would no longer be given since ballots could be decrypted and votes could be linked to voters. Therefore, everlasting BS is stronger, information-theoretic guarantee not offered by any REV systems relying on asymmetric cryptography [23].

Definition 3 *Receipt-Freeness. In a REV system guaranteeing RF, it is impossible for a voter to prove to a third-party (*e.g.*, a vote-buyer) how she*[1] *voted [67].*

Definition 4 *Coercion-Resistance. CR extends the notion of RF by ensuring the protection of the voter against forced abstention, randomized voting or (un-)willingly giving up voter credentials. CR is a stronger privacy guarantee than RF as it does not differentiate between honest and malicious voters [67].*

REV protocols in early work, such as [20], issued receipts upon casting a ballot, providing the voter the ability to trace its ballot on the bulletin board. It was quickly discovered [11] that this is flawed and facilitated vote-buying and coercion of the voter which brought about the notion of RF and CR. It is argued [67, 70] that in REV systems RF and CR are more important than in traditional voting due to the simplified scalability of vote-buying given digital receipts.

2.1.2 Verifiability

Verifiability is a major component in building and ensuring trust in any kind of voting system. In paper-based voting systems, the voter cannot verify that her ballot has been correctly included and counted as a part of the tallying process. She must trust the post, the government and the poll workers that each one performs their duties truthfully. In REV systems, the situation is even more problematic as physically observable processes (*e.g.*, mailing a ballot or anonymizing a ballot) are absent. Therefore, techniques are required that allow a voter to verify the correct and truthful execution of the voting protocol. The literature [67, 94] presents various forms of verifiability.

[1] In cryptography and REV literature, Alice and Bob are common fictional characters used to explain protocols or interactions. To keep in line with the standard, Alice the voter is female and thus referred to with her pronouns.

Definition 5 *Individual Verifiability (IV). In a REV system supporting IV, a voter can verify that her ballot has been included in the recorded set of ballots [94].*

Definition 6 *Universal Verifiability (UV). In a REV system supporting UV, anyone (*e.g.*, another voter or a third-party) can verify the outcome of the vote* i.e.*, that the recorded set of ballots corresponds with the result [70, 74].*

IV on its own is insufficient in guaranteeing verifiability in a REV system, as it is based on trusting other voters to verify that their cast ballots are included in the recorded set of ballots. To enable public audits of a vote in an efficient manner, UV is required [32, 94]. Alternatively, the literature defines a property similar to the combination of IV and UV but which additionally includes an assertion that the vote has been encrypted as intended, it is referred to as E2E verifiability [77].

Definition 7 *End-to-End Verifiability (E2E-V). A REV system supporting E2E-V is similar to IV and UV but instead composed of three parts [67, 74, 94]:*

- ***Cast-as-Intended (CaI)** verifiability enables a voter to verify that her intended vote is included in the ballot.*
- ***Recorded-as-Cast (RaC)** verifiability requires that a voter can verify that her ballot has been recorded without being altered.*
- ***Counted-as-Recorded (CaR)** verifiability means that the voter can verify that all valid, recorded votes have been correctly counted.*

CaI is an necessity in any REV system to remedy the problems of casting ballots in an uncontrolled environment and from uncontrolled devices (*e.g.*, mobile phone) [76]. Otherwise, it is possible for a malicious voting client to encrypt something else than the voter's choice and the voter would have no way of realizing this. On top of that, a malicious voting client could also encrypt multiple votes using the same randomness leading to identical ballots. Among other things to avoid clash attacks, *i.e.*, replacing identical ballots by an adversary, identical ballots are rejected by most REV systems [78].

2.1.3 Practical Properties

The literature defines additional properties which cannot be grouped into the two main categories [67, 70, 74, 94] and more practical. The most important ones are described below.

Definition 8 *Fairness. In a fair REV system, it is impossible for anyone to compute an intermediary result before the end of the voting period [50].*

Fairness is required to avoid that anyone can influence the outcome of a vote, by computing a partial tally and distributing it, while the vote is still running.

Definition 9 *Accountability. In an accountable REV system, misbehaving parties (*e.g.*, authorities) can be identified and excluded [76].*

E2E-V is insufficient in practice, REV systems also need to be accountable in order to guarantee the correct execution of the voting protocol by the involved parties. It is essential that misbehaving parties are held accountable (*e.g.*, via heavy fines or law suits) to maintain the trust and reputation of a REV system [76].

Definition 10 *Robustness. A REV system is considered robust if it is able to cope with unpredictable changes in and influences on its operation without impacting its availability or correct functioning [97].*

Definition 11 *Scalability. A REV system is considered scalable if an increased number of voters does not imposed any restrictions on the system or incur any unsustainable requirements on computational power or data storage.*

Robustness and scalability are practical properties necessary for the fault-tolerant, highly-available and safe operation of a REV system. Only if these properties are given can a REV system be applied in practice for large scale votes.

Definition 12 *Transparency. In a transparent REV systems, all processes are documented and publicly available. Each stakeholder has clearly defined roles and responsibilities. This also includes a security concept and protocols in case of an emergency or malfunctioning of the system (*e.g.*, attack, error) [35].*

2.1.4 Trust Assumptions & Trade-Offs

The difficulty of achieving the previously mentioned properties depends on the trust assumptions of the considered REV systems. If a trusted third party or stakeholder is acceptable, it is less demanding to achieve the desired properties. In theory, the ultimate goal is to have a REV system without any trust assumptions that is perfectly verifiable while everlastingly protecting the voter's privacy [67]. As likely assumed, is such a case impossible to achieve as privacy and verifiability have inherently contradicting goals. For example, it was shown [23] that UV and RF cannot simultaneously co-exist unless private communication channels are available between the stakeholders which is an unreasonable assumption. Or, that improving the level of BS, *i.e.*, disguising the relationship between the voter and its ballot, can actually lower the level of CR [77]. Therefore, the goal of REV research and system design is to strive for the most promising trade-offs *i.e.*, achieving sufficient privacy levels while retaining as much verifiability as possible [67].

In practice, the application of the REV system determines the acceptable trust assumptions and the necessary strength of the properties. Depending on the application, it can be viable to favour certain properties over others. For example, in case of national elections, the verifiability properties might be of greater importance than the RF property as large scale vote-buying is highly unlikely to go unnoticed. On the other hand, in a public

company's general assembly vote, the choices of major shareholders are usually publicly known and can be inferred from the outcome of the vote. To ensure the vote's outcome is as intended, a major shareholder might be tempted to buy votes. In such a case, RF can be more important than verifiability [23].

2.2 Distributed Ledger Technology

A distributed ledger (DL) can best be described as a public, distributed, append-only database storing all transactions executed on a network. In more technical terms, a DL is an immutable, backward-linked list formed by blocks of transactions. A block is a data structure that bundles a group of transactions. Transactions are generic in nature and allow the execution of arbitrary code (*e.g.*, money transfer, ownership proof, etc.) *i.e.*, allowing to mimic contractual relationships between parties. The DL forms an immutable chain as each block links to the preceding one by storing its cryptographic hash next to the transactions. Thereby, making it impossible to alter or remove agreed upon transactions contained in preceding blocks. Also, storing the hash of the preceding block reinforces the integrity as any change in a preceding block would change its hash and, therefore, change the hash of all following blocks. A DL is operated as a peer-to-peer (P2P) network and maintained by a set of peers following a consensus mechanism. This mechanism is a set of mathematical rules responsible for validating and synchronizing state changes without the need for a trusted authority. It allows all peers to agree on the integrity of any state change initiated by a transaction [86, 105, 108].

2.2.1 Permission Models & Consensus Mechanisms

Different types of DLs exist for different application purposes. Generally, they can be differentiated by their permission model as well as their consensus mechanism. In a permissionless setting, anyone is allowed to join the network as a an active (*e.g.*, a validator) or passive (read-only) participant at any point in time. There exists no fixed set of authorities that govern the network and consensus is reached for example, by "putting in the work". This refers to the commonly known Proof-of-Work (PoW) consensus mechanism that is based on participants attempting to solve a mathematical challenge and proving to a verifier that a certain amount of work, *i.e.*, computational power, has been performed. Well-known examples of permissionless setups and PoW are Bitcoin [84] and Ethereum [107]. On the other hand, in a permissioned setup, the set of authorities (and sometimes participants) is fixed. This work is based on a variant of permissioned DLs, namely public-permissioned DLs in which the set of authorities is fixed, reading is possible for anyone, new blocks are alternately produced in fixed time intervals and consensus is reached by a majority vote among the authorities. This mechanism is known as Proof-of-Authority (PoA) [5]. Examples of permissioned DLs are Hyperledger Fabric[2], Corda[3] or Parity Substrate[4]. As this work focuses on a setting in which a set of known,

[2]https://www.hyperledger.org/projects/fabric
[3]https://www.corda.net
[4]https://www.parity.io/substrate

semi-trusted authorities exist (*e.g.*, municipalities, cantons, government), the choice of a public-permissioned DL is evident.

2.2.2 DL as Public Bulletin Board

In Switzerland, it is required by law [32] for any REV system to provide verifiable evidence of its executed processes if the system shall be used by the majority of the public. An essential building block in achieving this verifiability and auditability requirement is the public bulletin board (PBB). A PBB is a publicly available audit trail of all storage and communication associated with a vote. Its storage is append-only to ensure the integrity. It provides a single and consistent representation of its state independent of the viewer and is backed by a distributed system to ensure reliability [67, 70]. Its main properties are [68]; public readability, immutability, integrity and redundancy. An attempt at a formal definition of the PBB is made in [60]. The authors propose an *interlinked* structure, based on a hash chain and similar in functionality to a DL. Also, write access is controlled via digital signature verification limited to a set of known public key addresses.

Given these requirements, public-permissioned DLs seem to be the perfect candidates for PBBs. The distribution of trust, *i.e.*, the ability to record and tally a vote, formalized through the consensus mechanism and the absence of an always online trusted third-party (TTP) are an advantage of DLs. By doing removing the TTP, a potential single point of failure is eliminated. Furthermore, to increase the trust in the REV system, authority roles could be assigned to other partially trusted organizations (*e.g.*, NGOs, vote observers, etc.) wanting to audit and safeguard the voting process [108]. On the other hand, the public readability also provides additional challenges given the desired privacy properties of REV systems. Since it is required that ballots are kept private and that no one can know if and how someone voted, cryptographic techniques such as encryption and shuffling ballots are required to ensure BS in public.

Chapter 3

Cryptography

The privacy and verifiability of Remote Electronic Voting (REV) systems is based on various cryptographic algorithms. These algorithms are composed of different underlying building blocks. This section introduces the most important building blocks required to understand the algorithms and concepts used in this work.

3.1 Hash Functions

A hash function H is used to generate a fixed-sized output y, called fingerprint, hash, or digest, from a variable size input x: $H(x) = y$. For a hash function H to be used in a cryptographic application, it must fulfil the following requirements [87, 101]:

- *Pre-Image Resistance (One-Way)*: Inverting the hash function is computationally infeasible, which means that it must be impossible to find x, for a given y, such that $H(x) = y$.
- *Second Pre-Image Resistance*: It must be impossible to find another input value x', where $x' \neq x$ but $H(x') = y$ and $H(x) = y$.
- *Collision Resistance*: Finding a pair of input values (x, z) such that $h(x) = h(z)$ must be impossible.

Hash functions must be efficient to compute for any given input and hard to invert. Also, finding a collision must be impossible to guarantee that the output cannot be from a different, potentially "fake", input. Given these properties, hash functions find a wide range of applications in cryptography, such as in message integrity checks, digital signatures, password storage or proof systems (*e.g.*, as a random oracle to make proofs non-interactive) [87].

3.2 Public Key Cryptography

Any public-key or asymmetric cryptography system is based on the intractability of a mathematical problem (*e.g.*, prime factorization of a large composite number, finding the discrete logarithm). It involves two distinct keys (public/private key) compared to symmetric cryptography. This is necessary since it must be impossible to infer the private key from the public key and/or the decrypted data from the encrypted data. The public key is used for encryption and digital signature verification, while the private key is used for decrypting and generating digital signatures [87, 101]. Any public-key system consists of the following components: a plaintext message m, an encryption algorithm E, a public/private key pair (pk, sk), an encrypted message (ciphertext) c and a decryption algorithm D [101]. Equation 3.1 illustrates the relation of the mentioned components.

$$m = D(c, sk) = D(E(m, pk), sk) \tag{3.1}$$

In practice, public-key cryptography is well researched and standards are established. Examples of commonly used public-key systems are RSA [93], Diffie-Hellmann Key Exchange [34] and ElGamal [37], described in Section 3.4.

3.3 Homomorphic Encryption

Homomorphic encryption is based on a structure-preserving mapping of two different operations. It allows for one operation to be performed on ciphertext, resulting in the same outcome as if the operation had been performed on plaintext. Thus, allowing for confidential information (*e.g.*, the content of a vote) to remain private while processing it in public [6, 43, 67]. The mapping can be defined as follows:

$$E(m_1, pk) \otimes E(m_2, pk) = E(m_1 \oplus m_2, pk) \tag{3.2}$$

As can be seen from equality (3.2), two ciphertexts can be combined into one without using the private key. The operators $\oplus$ and $\otimes$ are arbitrary and depend on the underlying cryptosystem. This property is the main concept behind REV systems based on homomorphic tallying and allows to compute the result of a vote (*i.e.*, summing all encrypted votes) without decrypting an individual ballot.

3.4 ElGamal Cryptosystem

The ElGamal cryptosystem is a homomorphic, probabilistic, public-key cryptosystem based on the intractability of the discrete logarithm problem in the cyclic group $\mathbb{G}$. The cyclic group $\mathbb{G}_q$ with prime order q is defined over a finite field $\mathbb{Z}_p$ which is defined over positive integers modulo p, where p is an odd prime. The cryptographic operations are

performed in a subgroup $\mathbb{G}_q \subset \mathbb{Z}_p^*$, a multiplicative cyclic group $\mathbb{G}_q$ defined over the finite field $\mathbb{Z}_p$. Alternatively, the cyclic group $\mathbb{G}$ can also be defined over an elliptic curve that is itself defined over a finite field [37].

3.4.1 Security

The security of the ElGamal cryptosystem is based on the mathematical hardness of finding discrete logarithms in cyclic groups, which is known as the decisional Diffie-Hellmann (DDH) assumption. The DDH assumptions is necessary to ensure that the ElGamal cryptosystem is secure against chosen plaintext attacks (`IND-CPA`) [13]. The most common group for which the DDH assumption is considered to hold is the subgroup of quadratic residues modulo a safe prime: $p = 2q + 1$ [14].

Definition 13 *(Decisional Diffie-Hellmann (DDH)) Given a multiplicative, cyclic group $\mathbb{G}_q$ of prime order q, and with generator g, the DDH assumption states that for any triple $(a, b, c) \in_r \mathbb{Z}_q^*$ (chosen uniformly, independently and at random from $\mathbb{Z}_q^*$), no efficient (polynomial time) algorithm can distinguish between g^{ab} and g^c (*i.e., *g^{ab} "looks" just like any other value in $\mathbb{G}$) [14].*

3.4.2 Key Generation, Encryption and Decryption

Key Generation. The public elements of the ElGamal crypto system are $\mathbb{G}, p, q$ and two independent generators g and h, where $g, h \in \mathbb{G}_q \setminus 1$. The private key sk is a random value: $sk \in_r \mathbb{Z}_q^*$, where $q = \frac{p-1}{2}$. The public key pk is defined as follows:

$$(sk, pk) = (sk, g^{sk} \mod p) \tag{3.3}$$

Encryption. A plaintext message m can be encrypted using a random value $r \in_r \mathbb{Z}_q^*$.

$$\begin{aligned} E(m, pk) = c &= (a, b) \\ &= (g^r \mod p, pk^r \cdot m \mod p) \end{aligned} \tag{3.4}$$

Decryption. A ciphertext $c = (a, b)$ is decrypted by computing the multiplicative inverse of a^{sk} multiplied with b.

$$\begin{aligned} D(c, sk) = m &= (a^{sk})^{-1} \cdot b \mod p \\ &= (g^{r^{sk}})^{-1} \cdot pk^r \cdot m \mod p \\ &= (g^{r^{sk}})^{-1} \cdot g^{sk^r} \cdot m \mod p \\ &= (g^{r \cdot sk})^{-1} \cdot g^{r \cdot sk} \cdot m \mod p \end{aligned} \tag{3.5}$$

3.4.3 Message Encoding

Messages in the ElGamal cryptosystem must be encoded, *i.e.*, mapped with a reversible function to a group element, such that the DDH assumption holds [25, 36]. Otherwise, the ElGamal cryptosystem is not secure against chosen plaintext attacks (`IND-CPA`). To accomplish this, different encodings with their own advantages and disadvantages exist, of which two are discussed in the following.

Variant 1: g^m

Exponent ElGamal message encoding has been introduced by Cramer et al. [25]. The idea is to encode any message $m \in \mathbb{Z}_q$ as g^m. The corresponding ciphertext is given by $E(m, pk) = c = (a, b) = (g^r, pk^r \cdot g^m)$. This type of encoding is used in Provotum 2.0 [72] and 3.0 [66] as it has the additional property of making the ElGamal cryptosystem *additive*, instead of *multiplicative* homomorphic. A downside of this encoding is that the plaintext, resulting from the decryption, must be decoded by computing the discrete logarithm of g^m (*i.e.*, $g^? = g^m$). Generally, computing discrete logarithms is hard. It is only possible for a restricted message space $m \in 0, 1$ where brute forcing or using the `baby-step-giant-step` algorithm is possible [36].

The example below visualizes how two ElGamal ciphertexts $E(m_1)$ and $E(m_2)$ can be homomorphically summed. The components are multiplied, *i.e.*, its exponents are summed, resulting in the same sum as if the plaintexts m_1 and m_2 had been summed.

$$E(m_1 + m_2, pk) = E(m_1, pk) \cdot E(m_2, pk) = (g^{r_1+r_2}, pk^{r_1+r_2} \cdot g^{m_1+m_2}) \tag{3.6}$$

$$m_1 + m_2 = \text{decode}(D(E(m_1 + m_2, pk), sk)) \tag{3.7}$$

Variant 2: $m^q \mod p \equiv 1$

An alternative method is to ensure all messages are restricted to be quadratic residues in a safe prime group $\mathbb{G}$ of order q. Using this approach, it is required to check that the following equality: $m^q \mod p \equiv 1$ holds before a message is encrypted. Otherwise, the message must not be encrypted [36]. This work makes use of the second variant as each ballot needs to be decrypted and no additional decoding step is required after the decryption. On the other hand, choosing the prior variant would be computationally infeasible. A downside of this encoding is that the equality check needs to be performed before encrypting, increasing the runtime due to the modulo exponentiation. Fortunately, the increase is negligible in practice for a single operation.

3.5 Re-Encryption

Re-encryption is a technique that can be used in probabilistic encryption systems to change the ciphertext without changing the respective plaintext. Since the ciphertext

depends on an independent random value r_1, its appearance can be altered by encrypting it again using a different independent random value r_2. The resulting ciphertext is equivalent as if it only depended on a single independent random value r_3. It is important to note that both pk and g need to remain the same throughout all encryptions [67].

generic re-encryption

$$E_{r_2}(E_{r_1}(m, pk), pk) = E_{r_3}(m, pk) \tag{3.8}$$

Re-encryption is used in REV systems to break the association between the voter and her ballot since it only changes its appearance but not its content. The re-encrypted ballot can no longer be associated with the voter while the content (*i.e.*, the vote) remains unchanged. In any probabilistic homomorphic encryption system, re-encryption is supported by homomorphically adding an encryption of 0 (3.9) or by multiplying with an encryption of 1 (3.10), depending on the type of message encoding, without changing the plaintext [67].

by addition

$$\begin{aligned} E_{r_1}(m) &= (g^{r_1}, pk^{r_1} \cdot g^m) \\ E_{r_2}(0) &= (g^{r_2}, pk^{r_2} \cdot g^0) \\ E_{r_1}(m) \cdot E_{r_2}(0) &= (g^{r_1} \cdot g^{r_2}, pk^{r_1} \cdot g^m \cdot pk^{r_2} \cdot g^0) = (g^{r_1+r_2}, pk^{r_1+r_2} \cdot g^m) \end{aligned} \tag{3.9}$$

by multiplication

$$\begin{aligned} E_{r_1}(m) &= (g^{r_1}, pk^{r_1} \cdot m) \\ E_{r_2}(1) &= (g^{r_2}, pk^{r_2} \cdot 1) \\ E_{r_1}(m) \cdot E_{r_2}(1) &= (g^{r_1} \cdot g^{r_2}, pk^{r_1} \cdot m \cdot pk^{r_2} \cdot 1) = (g^{r_1+r_2}, pk^{r_1+r_2} \cdot m) \end{aligned} \tag{3.10}$$

3.6 Mixnets

A mix-net is a strategy of preserving the privacy of the voters by ensuring that the ballots, *i.e.*, the ciphertexts, cannot be associated with the voting choice. If a REV system would simply decrypt the ballots once the voting period has ended, the system could not guarantee ballot privacy. Anyone that can link a ciphertext to a voter, would be able to link it to the corresponding plaintext. Instead, by using a mixnet to irreversibly shuffle the encrypted ballots, the linking between the cipher- and plaintext can be prevented. A mixnet changes the appearance and order of the ciphertexts such that its inputs cannot be linked trivially to its outputs. Important hereby is, that the mixnet must maintain the integrity of the ballots. It must not be possible that the mixnet adds or removes ballots or changes the content of a ballot. Thus, a proof of a correct shuffle is required. This can be done using both *re-encryption mixnets* or *decryption mixnets* [58, 67, 70, 98].

3.6.1 Decryption Mixnets

Decryption mixnets have been first introduced by Chaum [19]. A *decryption mix* works by using several layers of encryption. For each authority (mixer), the vote is encrypted with its public key. During every step of the mixnet, the ballots are stripped off a different encryption layer resulting in a different shape and output. The results of the intermediate steps are still ciphertexts and, therefore, do not reveal any information about the votes. Only the sequential decryption of all encryptions can reveal the plaintexts and in turn the result of the vote. The approach is similar to the technique used in anonymous routing such as in a TOR[1] network [58, 67, 70].

3.6.2 Re-Encryption Mixnets

Re-encryption mixnets have been first introduced by Park et al. [88]. They require a probabilistic public key encryption system such as ElGamal [37] that allows to re-encrypt a ciphertext (see Section 3.5). The process of a re-encryption mixnet can be compared with the physical task of shaking a ballot box to shuffle the ballots and, therefore, making it impossible to associate a ballot with a specific voter [52, 55]. More formally, a re-encryption mixnet is formed by a series of cryptographic shuffles, each composed of a secret permutation and a re-encryption function. The cryptographic shuffle re-encrypts and permutes a group of ballots *i.e.*, changes the ciphertext appearance and outputs the result in a randomized order. To avoid recoverability, this step can be repeated multiple times [58, 61, 67, 70]. The algorithms used in this work are based on the CHVote specification [52][2].

GeneratePermutation

known: p, q, g
input: permutation size N
$I = \langle 1, ..., N \rangle$
for $i = 0, ..., N-1$ **do**
  $r \in_r [i, N-1]$
  $j_{i+1} = I[r]$
  $I[r] = I[i]$
output: $\psi = (j_1, ..., j_N)$

Figure 3.1: Pseudo-Code Permutation Algorithm

Definition 14 *(Permutation) A list of messages $M = (m_1, m_2, ..., m_n)$ is considered permuted if the positions of all its entries are randomly reassigned $M' = (m_5, m_n, m_1, ..., m_2)$ while keeping the same number of elements. Also, none of the elements is allowed to*

[1] https://www.torproject.org

[2] The CHVote specification is available online: `https://eprint.iacr.org/2017/325.pdf`. The implemented algorithms 8.42 *GenShuffle* and all helper algorithms used within 8.42.

change. A permutation is defined as a mapping $\psi : \{1, ..., N\} \rightarrow \{1, ..., N\}$ *and can be selected at random from the set of all possible permutations using the Knuth (Fisher-Yates) algorithm [52, 55]. A pseudo-code version of the algorithm is illustrated in Figure 3.1.*

Definition 15 *(Cryptographic Shuffle) A cryptographic shuffle transforms a list of ciphertexts* $\boldsymbol{c} = (c_1, c_2, ..., c_n)$ *into* $\boldsymbol{c'} = (c'_n, c'_2, ..., c'_1)$ *using permutation and re-encryption such that* $c'_j = \texttt{re-encrypt}(c_i, pk), \forall j = \psi(i)$ *[52, 55]. A pseudo-code version of the algorithm is found in Figure 3.2.*

GenerateShuffle

known: p, q, g

inputs:

– encryptions $\mathbf{c} = (c_1, ..., c_N)$

– public key pk

$\psi = (j_1, ..., j_N) \in_r \Psi_N$ (see 3.1)

for $i = 1, ..., N$ **do**

$j_i = \psi(i)$

$c'_i = \texttt{re-encrypt}(c_{j_i}, pk)$

output: $\mathbf{c'} = (c'_1, ..., c'_N)$

Figure 3.2: Pseudo-Code Shuffle Algorithm

To recap, a multiplicative homomorphic encryption $c = E_{r_1}(m, pk)$ can be re-encrypted as $c' = \texttt{re-encrypt}(c, pk) = E_{r_2}(c, pk) = c \cdot E_{r_2}(0, pk)$ as shown in Section 3.5. The output $\mathbf{c'}$ represents the same plaintexts $(m_1, m_2, ..., m_n)$ as $\mathbf{c}$ but their ciphertext representation is different and in permuted order [52, 55]. The correctness of the cryptographic shuffle needs to be proven for each re-encrypted message. This is called a *proof of shuffle* and is explained in detail in Section 3.8.4.

3.6.3 Summary

In both variants (decryption & re-encryption mixnet), the outcome is privacy preserving as an adversary cannot identify voters using their ballots. Nevertheless, re-encryption mixnets have become the de-facto standard for REV systems leveraging mixnet techniques. This is because mixnets in REV systems come into play once all ballots have been cast and the voters' signatures have been verified. A re-encryption mixnet is advantageous in this use case as the voters only need to perform a single encryption while the shuffling process can be repeated multiple times independently of the voters' original encryptions [52, 58, 55, 61, 67, 70].

3.7 Multi-Party Computation

Multi-Party Computation (MPC) is a cryptographic technique with which a group of participants can compute a shared function without revealing their private inputs [90]. In REV systems, the technique is used to distribute trust equally among all participants avoiding single points of failure. For example, to avoid that a single entity is able to decrypt individual ballots, a distributed key generation (DKG) scheme can be used.

3.7.1 Distributed Key Generation

A DKG protocol allows a set of participants to generate a collective secret with its shares spread among the participants. In REV systems, such a protocol is used to generate the vote's public key. Each participant generates a (public/private) key pair locally and publishes their public key share. All public key shares are then combined into the common public key. Only a collective effort can retrieve the private key, as the individual shares are not publicly known, while any subset smaller than the number of participants cannot [46, 51, 67, 71, 98]. The protocol used in this work is based on [25, 90] and requires a probabilistic, public-key cryptosystem (*e.g.*, ElGamal) in which all n participants present during the key generation also need to be present during the decryption. In a practical scenario, the participants of the protocol should therefore be a set of trustworthy parties (*e.g.*, the cantons).

Key Pair Generation. Each participant $i \in [1, n]$ creates a (public/private) key pair (pk_i, sk_i), as shown in equation 3.11, resulting in n key pairs. The private key shares sk_i are kept secret, while the public key shares pk_i are published and combined into the vote's public key (3.12). The vote's public key pk is then used by the voters to encrypt their ballots. Since this process occurs offline and cannot be verified, its correct execution needs to be proven similarly to the shuffle proof required as a part of the mixing process. In this case, a proof of knowledge of a private key share sk_i that belongs to a public key share pk_i is required which is explained in detail in Section 3.8.1.

$$(pk_i, sk_i) = (g^{r_i} \mod p, r_i) \tag{3.11}$$

$$pk = \prod_{i=1}^{n} pk_i \mod p \tag{3.12}$$

3.7.2 Cooperative Decryption

Similar to the DKG protocol, each participant i must generate a set of decryption shares d_{ij} such that the ballots, encrypted with the vote's public key, can be decrypted. It is necessary that all and the same n participants which took part in the DKG protocol also

take part in the decryption protocol. The decryption protocol is composed of three steps which can be grouped into an offline (Step 1) and online part (Step 2 & 3).

Step 1 (Equation 3.13). Each participant i partially decrypts the ballots c, using its private key sk_i, producing a set of decryption shares d_i. These shares do not reveal anything about the plaintexts. Since accesses to the private key sk_i is required in this step, the operation must occur offline and be accompanied by a proof (3.8.2) similar to the one performed as part of the DKG protocol [52].

$$\begin{aligned} \textbf{input} \quad & \text{ciphertexts: } \mathbf{c} = [c_1, c_2, ..., c_j, ..., c_m], c_j = (a, b) \\ \textbf{output} \quad & \text{partial decryptions: } d_i = [d_1, d_2, ..., d_j, ..., d_m] \\ & d_j = a_j^{sk_i} \mod p \\ & = g^{r_j \cdot sk_i} \mod p \end{aligned} \tag{3.13}$$

Step 2 (Equation 3.14). The decryption shares of all participants are combined to retrieve the combined decryption shares [52]. This step can be performed online and, therefore, can be verified by anyone, thus not requiring an proof.

$$\begin{aligned} \textbf{input} \quad & \text{partial decryptions: } \mathbf{D} = (d_{ij}) \in \mathbb{G}_q^{N \times M} \\ \textbf{output} \quad & \text{combined partial decryptions: } d = [d_1, d_2, ..., d_j, ..., d_m] \\ & d_j = \prod_{i=1}^{N} d_{ij} \mod p \end{aligned} \tag{3.14}$$

Step 3 (Equation 3.15). The combined decryption shares are used to retrieve the decrypted ballots [52]. Depending on the message encoding (3.4.3), the decrypted ballots might still have to be decoded to reveal the plaintexts [52]. Again, this step can be performed online and is therefore publicly verifiable for anyone.

$$\begin{aligned} \textbf{inputs} \quad & \text{ciphertexts: } \mathbf{c} = [c_1, c_2, ..., c_j, ..., c_m], c_j = (a, b) \\ & \text{combined partial decryptions: } d = [d_1, d_2, ..., d_j, ..., d_m] \\ \textbf{output} \quad & \text{plaintexts: } m = [m_1, m_2, ..., m_j, ..., m_m] \\ & m_j = \frac{b_j}{d_j} \mod p \\ & = pk^{r_j} \cdot m_j \cdot (g^{r_j sk})^{-1} \mod p \end{aligned} \tag{3.15}$$

3.8 Zero-Knowledge Proofs

Zero-Knowledge Proofs (ZKP) are a cryptographic technique that allows a prover P to prove to a verifier V that it knows about a secret s without revealing anything about s. More generally, a ZKP is used to prove some boolean statement over publicly and privately known arguments, without revealing the private arguments. For example, in REV systems, a ZKP is used to prove knowledge of a private key that belongs to a certain public key to assert that the private key was in a given operation [67, 98].

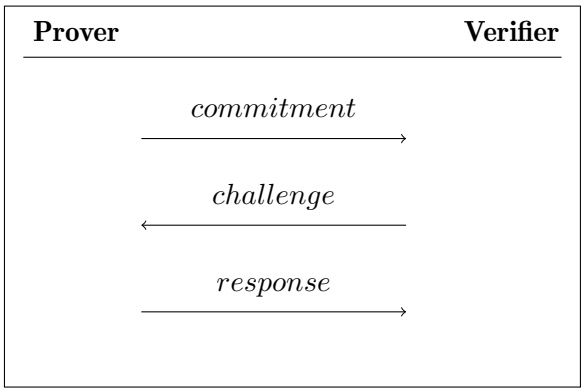

Figure 3.3: A generic Σ protocol

Interactive ZKP. A common way to model ZKPs is to use a three-move interactive protocol between the prover P and the verifier V. As visualized in Figure 3.3, P initially commits to an outcome based on its private knowledge, then V challenges P and thus P needs to reveal a part of its commit to assert consistency with its claim. The repeated execution of this commit-challenge-response scheme allows V to gain confidence in the proof [67, 98]. The protocol is also commonly known as Σ-protocol, named after Cramer [24], and can be defined as follows.

Definition 16 *(Σ Protocol.) Σ-protocols must be (1) complete, (2) sound and (3) zero-knowledge. (1) For a true statement, a honest prover P must always succeed in convincing an honest verifier V. (2) For a false statement, a honest verifier V will only be convinced with negligible probability, independent of the prover's doing. (3) It must not be possible for anyone to learn anything from the proof, apart for its correctness [24].*

Non-Interactive ZKP (NIZKP). As interactive ZKPs require more than one communication step between P and V, which is time-consuming and costly, it can make their application in REV systems infeasible. Fortunately, interactive ZKPs can be made non-interactive using the Fiat-Shamir [42] transformation. The idea is to replace the interactively obtained challenge from V by an unpredictable challenge generated by P. The challenge is generated using a cryptographic hash function, serving as a local random oracle [42]. Both the commitments as well as the public arguments or statement to be proven need to be hashed, to avoid the pitfall of using a weak Fiat-Shamir transformation (only hashing the commitments) which leads to unsound proofs in the case of a dishonest

prover [12]. The instantiations of the NIZKPs used in this work are introduced in the next sections.

3.8.1 Key Generation Proof

To show knowledge of a private key $sk \in_r \mathbb{Z}_q^*$ that belongs to a public key $pk = g^{sk}$, the Schnorr Proof [95] can be used. It is also known as pre-image proof and is a proof of knowledge of a discrete logarithm of $sk = \log_g(g^{sk})$ [38]. It is generated and verified as shown in Figure 3.4, requiring the previously generated public/private key pair (pk, sk) (Section 3.3) as input.

Prover		**Verifier**
knows id, sk, pk, g, p, q		knows id, pk, g, p, q
commitments		
$a \in_r \mathbb{Z}_q^*$		
$b = g^a \mod p$		
challenge		
$c = \texttt{hash}(id, pk, b) \mod q$		
$d = a + c \cdot sk \mod q$		
	$\xrightarrow{(c,d)}$	**re-compute**
		$b = \frac{g^d}{pk^c} \mod p$
		$c' = \texttt{hash}(id, pk, b) \mod q$
		verify
		$c \stackrel{!}{=} c'$
		$g^d \stackrel{!}{=} b \cdot pk^{c'} \mod p$

Figure 3.4: Key Generation Proof

3.8.2 Decryption Proof

To prove that a partial decryption of a ciphertext $c = (a, b)$ is correct, the following needs to be proven: $d = a^{sk_i}$, where $pk_i = g^{sk_i}$. The proof used in this work (Figure 3.5) is inspired by the CHVote specification [52] and is a modified version of the Chaum-Pedersen proof [22] such that a single proof can show the correct partial decryption for a batch of ciphertexts. It is a proof of discrete logarithm equality of $\log_g(g^r) = \log_h(h^r)$ and is similar to the Schnorr Proof [95].

Prover		**Verifier**
knows id, sk_i, pk_i, g, p, q		knows id, pk_i, g, p, q
knows $c_i = (a_i, b_i) \in \mathbb{Z}_p^2, d_i \in \mathbb{Z}_p$		knows $c_i = (a_i, b_i) \in \mathbb{Z}_p^2, d_i \in \mathbb{Z}_p$
inputs		**inputs**
$\mathbf{c} = (c_1, c_2, ..., c_N)$		$\mathbf{c} = (c_1, c_2, ..., c_N)$
$\mathbf{d} = (d_1, d_2, ..., d_N)$		$\mathbf{d} = (d_1, d_2, ..., d_N)$
commitments		
$w \in_r \mathbb{Z}_q^*$		
$t_0 = g^w \mod p$		
$t_i \leftarrow a_i^w \mod p$		
$\mathbf{t} = (t_0, t_1, ..., t_N)$		
challenge		
$c = \texttt{hash}(id, pk_i, \mathbf{c}, \mathbf{d}, \mathbf{t}) \mod q$		
$s = w - c \cdot sk_i \mod q$		
	$\xrightarrow{(c, s)}$	**re-compute**
		$t_0 = {pk_i}^c \cdot g^s \mod p$
		$t_i \leftarrow d_i^c \cdot a_i^s \mod p$
		$\mathbf{t} = (t_0, t_1, ..., t_N)$
		$c' = \texttt{hash}(id, pk_i, \mathbf{c}, \mathbf{d}, \mathbf{t}) \mod q$
		verify
		$c \stackrel{!}{=} c'$

Figure 3.5: Decryption Proof

3.8.3 Re-Encryption Proof

A regular non-interactive ZKP (NIZKP) is transferable, meaning that the proof's statement can be verified by anyone. In the context of REV systems, this is not always desireable. For example, using a regular NIZKP a voter would be able to prove to a vote buyer how she voted by simply forwarding the proof. To mitigate this risk, a NIZKP can be made non-transferable by adapting such that it only convinces a designated verifier. A designated verifier proof claims a statement similar to: *knowledge of V's private key OR knowledge of the secret*. And, since only the designated verifier V has knowledge of the private key, is thus able to check if the claim is correct or not [67, 98].

In this work, a designated verifier proof is used by the randomizer to prove to a voter that an encrypted ballot c' is a valid re-encryption of the voter's original encrypted ballot c. The idea is to re-encrypt with the identity element, *i.e.*, 0 (addition) and 1 (multiplication), to change the appearance of the ciphertext but not the content of the ballot. The re-encryption variants can be seen in (3.9) and (3.10). The proof protocol shown in Figure 3.6 is inspired by Hirt *et al.* [65] and makes use of the Schnorr proof [95]. It has been adapted for the use with non-encoded, multiplicative homomorphic encryptions (see

Section 3.4.3). The proof is a logical OR-combination of two parts. A proof of knowledge of a discrete logarithm such that $sk_{voter} = \log_g(pk_{voter}) = \log_g(g^{sk_{voter}})$. And, a proof such that $c_1 \overset{!}{=} c' \ominus c \overset{!}{=} c' \oplus c^{-1}$ is satisfied, given $c = E(m, r_0)$ (the voter's initial encryption), $c_1 = E(1, r_1)$ (an encryption of 1 using the re-encryption random r_1) and $c' = \texttt{re-encrypt}(c, r_1) = c_1 \oplus c$ (the randomizer's re-encryption of c). Referring back to the statement to be proven, the *secret* is the random value used in the re-encryption r_1.

Prover		**Verifier**
knows pk_v, g, p, q, r_1, c_1		knows sk_v, pk_v, g, p, q, c_1
$c_1 = E(1, r_1) = (g^{r_1}, pk_v^{r_1} \cdot 1)$		$pk_v = g^{sk_v}$
commitments		
$r_2, h_2, s_2 \in_r \mathbb{Z}_q^*$		
$c_1' = E(1, r_2) = (g^{r_2}, pk_v^{r_2} \cdot 1)$		
$t_2 = g^{s_2} \cdot pk_v^{-h_2} \mod p$		
challenge		
$h = \texttt{hash}(c_1, c_1', t_2) \mod q$		
$h_1 = h - h_2 \mod q$		
$c = h_1 \cdot r_1 + r_2 \mod q$		
	$\xrightarrow{c, c_1', h_1, h_2, s_2, t_2}$	**re-compute**
		$h = \texttt{hash}(c_1, c_1', t_2) \mod q$
		verify
		$h \overset{!}{=} h_1 + h_2 \mod q$
		$E(1, c) \overset{!}{=} h_1 \cdot c_1 \oplus c_1'$
		$g^{s_2} \overset{!}{=} pk_v^{h_2} \cdot t_2 \mod p$

Figure 3.6: Designated Verifier Re-Encryption Proof

3.8.4 Shuffle Proof

The type of shuffle proof used in this work has been first introduced by Furukawa and Sako [45] and is based on permutation matrices. The exact implementation is a proof of a correct shuffle [61, 104] and adapted from the CHVote specification [52]. In the following, the necessary concepts are introduced to understand how the proof works and what it is composed of. The algorithms[3] implemented for the proof generation and verification can be seen in Figures 3.7 and 3.8. The auxiliary algorithms referenced in both the generation and verification are depicted in Figures 3.9, 3.10, 3.11 and 3.12.

[3]The CHVote specification is available online: `https://eprint.iacr.org/2017/325.pdf`. The algorithms implemented are 8.45 *GenShuffleProof* and 8.46 *VerifyShuffleProof*, including all helpers mentioned in either one of the two.

As briefly touch upon in Section 3.6.2, the correctness of a cryptographic shuffle, which transforms a list of ciphertexts $\mathbf{c} = (c_1, c_2, ..., c_n)$ into $\mathbf{c'} = (c'_n, c'_2, ..., c'_1)$ using permutation and re-encryption such that $c'_j = \texttt{re-encrypt}(c_i, r'_i), \forall j = \psi(i)$, needs to be proven for each re-encryption. This can be done by proving knowledge of the permutation ψ and the vector of random factors used for re-encryption $\mathbf{r'} = (r'_1, r'_2, ..., r'_n)$ [52, 55]. The proof looks as follows:

$$\pi_\psi = \text{NIZKP}[(\psi, R') : c_j = \texttt{shuffle}(c_i, r'_i, \psi), \forall j = \psi_i(i)] \tag{3.16}$$

The result is a triple $(\mathbf{c}, \mathbf{c'}, \pi_\psi)$ containing the initial list of ciphertexts $\mathbf{c}$, its shuffled (*i.e.*, re-encrypted and permuted) transformation $\mathbf{c'}$ and the proof attesting the validity of the shuffle. Unfortunately, standard pre-image proofs such as the Schnorr proof 3.8.1, cannot be applied to the cryptographic shuffle as it is not homomorphic, which is a requirement to make the proof non-interactive [55]. For such a proof to be applicable in a REV system, it needs to be possible to make it non-interactive as performing multiple interactions between the PBB and every validator is impractical. Therefore, an alternative homomorphic version of the proof is required. This is an active area of research with many different approaches and strategies being explored. One of the most widely explored alternative proofs has been proposed by Wikström and Terelius [104]. It is the basis for this work and composed of an online and offline part which are discussed in the following.

Definition 17 *(Pedersen Commitment) A protocol that allows a prover P to commit to a secret value m without revealing it or being able to change it later on: $c = Commit(m, r) = g^m h^r \mod p$. h and g are independent generators of G_q, the same subgroup of Z_p^* ($p = 2q+1$) that the ElGamal encryption system uses. The commitment c can be publicly stored without revealing any information about the secret value m. Once P reveals m and r, any verifier V can check that $c = Commit(m, r) = g^m h^r \mod p$.*

In the first part (offline), the prover P computes a Pedersen commitment to a permutation matrix: $c = Commit(\psi, \mathbf{r})$, where the value of the commitment is the permutation function ψ [55, 104].

Definition 18 *(Permutation Matrix) A permutation matrix $B_\psi = (b_{ij})_{N \times N}$ fulfils the following two conditions: (1) the elements of each row must sum up to one: $\sum_{j=1}^{N} b_{ij} = 1$ and (2) each row contains exactly one non-zero element: $\prod_{i=1}^{N} \sum_{j=1}^{N} b_{ij} x_i = \prod_{i=1}^{N} x_i$. Therefore, the permutation matrix is composed of rows with a single one while all other entries are zeros [55].*

In the second part (online), the prover P shows that the permutation matrix is used in the cryptographic shuffle to transform $\mathbf{c}$ into $\mathbf{c'}$ using the standard pre-image proof. Now, one must show in zero-knowledge that the two conditions introduced in definition 18 are satisfied for B_ψ. Together, with the commitment from the offline part and the proof for correct re-encryption, one has created a zero-knowledge proof for a cryptographic shuffle without revealing any information about the permutation ψ nor the re-encryption random values $\mathbf{r'}$ [55, 104].

GenerateShuffleProof

known: p, q, g, h

inputs:

- encryptions $\mathbf{e} = (e_1, ..., e_N), e_i = (a_i, b_i) = (g^{r_i}, pk^{r_i} \cdot m)$
- shuffled encrpytions $\tilde{\mathbf{e}} = (\tilde{e}_1, ..., \tilde{e}_N)$
- re-encryption randoms $\tilde{\mathbf{r}} = (\tilde{r}_1, ..., \tilde{r}_N) \in \mathbb{Z}_q^N$
- permutation $\psi = (j_1, ..., j_N) \in \Psi_N$
- public key pk
- vote identifier id_{vote}

$\mathbf{h} = \textbf{GetIndependentGenerators}(N, id_{vote}), \mathbf{h} = (h_1, ..., h_N), h_i \in \{\mathbb{Z}_p \setminus 1\}$

$(\mathbf{c}, \mathbf{r}) = \textbf{GeneratePermutationCommitments}(\psi, \mathbf{h})$

$\mathbf{c} = (c_1, ..., c_N), c_i \in \mathbb{Z}_p, \mathbf{r} = (r_1, ..., r_N), r_i \in \mathbb{Z}_q$

$\mathbf{u} = \textbf{GetChallenges}(N, \mathbf{e}, \tilde{\mathbf{e}}, \mathbf{c}, pk), \mathbf{u} = (u_1, ..., u_N), u_i \in \mathbb{Z}_q$

$\tilde{\mathbf{u}} = (\tilde{u}_1, ..., \tilde{u}_N), \tilde{u}_i \leftarrow u_{j_i}$

$(\hat{\mathbf{c}}, \hat{\mathbf{r}}) = \textbf{GenerateCommitmentChain}(\tilde{\mathbf{u}})$

$\hat{\mathbf{c}} = (\hat{c}_1, ..., \hat{c}_N), \hat{c}_i \in \mathbb{Z}_p, \hat{\mathbf{r}} = (\hat{r}_1, ..., \hat{r}_N), \hat{r}_i \in \mathbb{Z}_q$

$R_0 = 0, U_0 = 1$

for $i = 1, ..., N$ **do**

$\hat{w}_i, \tilde{w}_i \in_r \mathbb{Z}_q$

$R_i = \hat{r}_i + \tilde{u}_i \cdot R_{i-1} \mod q, R'_i = \hat{w}_i + \tilde{w}_i \cdot R_{i-1} \mod q$

$U_i = \tilde{u}_i \cdot U_{i-1} \mod q, U'_i = \tilde{w}_i \cdot U_{i-1} \mod q$

$\hat{t}_i = g^{R'_i} \cdot h^{U'_i} \mod q$

$w_1, w_2, w_3, w_4 \in_r \mathbb{Z}_q$

$t_1 = g^{w_1} \mod p$

$t_2 = g^{w_2} \mod p$

$$t_3 = g^{w_3} \prod_{i=1}^{N} h_i^{\tilde{w}_i} \mod p$$

$$(t_{4,1}, t_{4,2}) = (pk^{-w_4} \prod_{i=1}^{N} \tilde{b}_i^{\tilde{w}_i} \mod p, g^{-w_4} \prod_{i=1}^{N} \tilde{a}_i^{\tilde{w}_i} \mod p)$$

$c = \texttt{hash}(\mathbf{e}, \tilde{\mathbf{e}}, \mathbf{c}, \hat{\mathbf{c}}, pk, t_1, t_2, t_3, t_{4,1}, t_{4,2}, (\hat{t}_i, ..., \hat{t}_N)) \mod q$

$\mathbf{v} = (v_1, ..., v_N), v_N = 1, v_{i-1} \leftarrow \tilde{u}_i \cdot v_i \mod q$

$$\bar{r} = \sum_{i=1}^{N} r_i \mod q, \hat{r} = \sum_{i=1}^{N} \hat{r} \cdot v_i \mod q, r = \sum_{i=1}^{N} r_i \cdot u_i \mod q, \tilde{r} = \sum_{i=1}^{N} \tilde{r}_i \cdot u_i \mod q$$

$s_1 = w_1 - c \cdot \bar{r} \mod q$

$s_2 = w_2 - c \cdot \hat{r} \mod q$

$s_3 = w_3 - c \cdot r \mod q$

$s_4 = w_4 - c \cdot \tilde{r} \mod q$

$\hat{\mathbf{s}} = (\hat{s}_1, ..., \hat{s}_N), \hat{s}_i \leftarrow \hat{w}_i - c \cdot \hat{r}_i \mod q$

$\tilde{\mathbf{s}} = (\tilde{s}_1, ..., \tilde{s}_N), \tilde{s}_i \leftarrow \tilde{w}_i - c \cdot \tilde{u}_i \mod q$

output: $\pi = (c, s_1, s_2, s_3, s_4, \hat{\mathbf{s}}, \tilde{\mathbf{s}}, \mathbf{c}, \hat{\mathbf{c}})$

Figure 3.7: Shuffle Proof Generation

VerifyShuffleProof

known: p, q, g, h

inputs:

- shuffle proof $\pi = (c, s_1, s_2, s_3, s_4, \hat{\mathbf{s}}, \tilde{\mathbf{s}}, \mathbf{c}, \hat{\mathbf{c}})$
- encryptions $\mathbf{e} = (e_1, ..., e_N), e_i = (a_i, b_i) = (g^{r_i}, pk^{r_i} \cdot m)$
- shuffled encrpytions $\tilde{\mathbf{e}} = (\tilde{e}_1, ..., \tilde{e}_N)$
- public key pk
- vote identifier id_{vote}

$\mathbf{h} = \textbf{GetIndependentGenerators}(N, id_{vote}), \mathbf{h} = (h_1, ..., h_N), h_i \in \{\mathbb{Z}_p \setminus 1\}$

$\mathbf{u} = \textbf{GetChallenges}(N, \mathbf{e}, \tilde{\mathbf{e}}, \mathbf{c}, pk), \mathbf{u} = (u_1, ..., u_N), u_i \in \mathbb{Z}_q$

$\hat{c}_0 = h$

$$\bar{c} = \prod_{i=1}^{N} c_i \div \prod_{i=1}^{N} h_i \mod p$$

$$u = \prod_{i=1}^{N} u_i \mod q$$

$$\hat{c} = \hat{c}_N \div h^u \mod p$$

$$\tilde{c} = \prod_{i=1}^{N} c_i^{u_i} \mod p$$

$$(\tilde{a}, \tilde{b}) = (\prod_{i=1}^{N} a_i^{u_i} \mod p, \prod_{i=1}^{N} b_i^{u_i} \mod p)$$

$$\hat{\mathbf{t}} = (\hat{t}_1, ..., \hat{t}_N), \hat{t}_i \leftarrow \hat{c}_i^{c} \cdot g^{\hat{s}_i} \cdot \hat{c}_{i-1}^{\tilde{s}_i} \mod p$$

$$t_1 = \bar{c}^{c} \cdot g^{s_1} \mod p$$

$$t_2 = \hat{c}^{c} \cdot g^{s_2} \mod p$$

$$t_3 = \tilde{c}^{c} \cdot g^{s_3} \prod_{i=1}^{N} h_i^{\tilde{s}_i} \mod p$$

$$(t_{4,1}, t_{4,2}) = (\tilde{b}^{c} \cdot pk^{-s_4} \prod_{i=1}^{N} \tilde{b}_i^{\tilde{s}_i} \mod p, \tilde{a}^{c} \cdot g^{-s_4} \prod_{i=1}^{N} \tilde{a}_i^{\tilde{s}_i} \mod p)$$

$$c' = \texttt{hash}(\mathbf{e}, \tilde{\mathbf{e}}, \mathbf{c}, \hat{\mathbf{c}}, pk, t_1, t_2, t_3, t_{4,1}, t_{4,2}, (\hat{t}_i, ..., \hat{t}_N)) \mod q$$

output: $c \equiv c'$

Figure 3.8: Shuffle Proof Verification

GetIndependentGenerators

known: p, q, g
inputs:
– number of independent generators $n \in \mathbb{N}$
– vote identifier id_{vote}
for $i = 1, ..., N$ **do**
$x = 0$
repeat
$x = x + 1$
$h_i = \texttt{hash}(id_{vote}, \text{“const”}, i, x) \mod p$
$h_i = h_i^2 \mod p$
until $h_i \notin \{0, 1\}$
output: $\mathbf{h} = (h_1, ...h_N), h_i \in \{\mathbb{Z}_p \setminus 1\}$

Figure 3.9: Algorithm to retrieve n independent generators for vote id_{vote}.

GeneratePermutationCommitment

known: p, q, g
inputs:
– permutation $\psi = (j_1, ..., j_N) \in \Psi_N$
– Independent Generators $\mathbf{h} = (h_1, ..., h_N), h_i \in \{\mathbb{Z}_p \setminus 1\}$
for $i = 1, ..., N$ **do**
$r_{j_i} \in_r \mathbb{Z}_q$
$c_{j_i} = g^{r_{j_i}} \cdot h_i \mod p$
$\mathbf{r} = (r_1, ..., r_N), r_i \in \mathbb{Z}_q$
$\mathbf{c} = (c_1, ..., c_N), c_i \in \mathbb{Z}_p$
output: $(\mathbf{c}, \mathbf{r})$

Figure 3.10: Algorithm to generate permutation commitment for permutation ψ.

GetChallenges

known: p, q, g

inputs:

- number of independent generators $n \in \mathbb{N}$
- encryptions $\mathbf{e} = (e_1, ..., e_N), e_i = (a_i, b_i) = (g^{r_i}, pk^{r_i} \cdot m)$
- shuffled encrpytions $\tilde{\mathbf{e}} = (\tilde{e}_1, ..., \tilde{e}_N)$
- permutation commitments $\mathbf{c} = (c_1, ..., c_N), c_i \in \mathbb{Z}_p$
- public key pk

$H = \texttt{hash}(\mathbf{e}, \tilde{\mathbf{e}}, \mathbf{c}, pk)$

for $i = 1, ..., N$ **do**

$I = \texttt{hash}(i)$

$u_i = \texttt{hash}(H, I) \mod 2^\tau, \tau = 128$

output: $\mathbf{u} = (u_1, ..., u_N), u_i \in \mathbb{Z}_q$

Figure 3.11: Algorithm to generate n challenge values for a set of public input values $\mathbf{e}, \tilde{\mathbf{e}}, \mathbf{c}, pk$.

GenerateCommitmentChain

known: p, q, g, h

input: permuted public challenges $\tilde{\mathbf{u}} = (\tilde{u}_1, ..., \tilde{u}_N), \tilde{u}_i \in \mathbb{Z}_q$

$R_0 = 0, U_0 = 1$

for $i = 1, ..., N$ **do**

$\hat{r}_i \in_r \mathbb{Z}_q$

$R_i = \hat{r}_i + \tilde{u}_i \cdot R_{i-1} \mod q, U_i = \tilde{u}_i \cdot U_{i-1} \mod q$

$\hat{c}_i = g^{R_i} \cdot h^{U_i} \mod p$

$\hat{\mathbf{r}} = (\hat{r}_1, ..., \hat{r}_N), \hat{r}_i \in \mathbb{Z}_q$

$\hat{\mathbf{c}} = (\hat{c}_1, ..., \hat{c}_N), \hat{c}_i \in \mathbb{Z}_p$

output: $(\hat{\mathbf{c}}, \hat{\mathbf{r}})$

Figure 3.12: Algorithm to generate commitment chain for the permuted public challenges $\tilde{\mathbf{u}}$.

Chapter 4

Related Work

This section presents different research efforts made in the area of remote electronic voting (REV). REV research has been initiated more than 40 years ago by David Chaum with his work on untracable electronic messaging [19]. It is the first known proposal and based on encrypted ballots and decryption mixes. Since then, a plethora of efforts have been made in different directions summarized and discussed in various survey papers [2, 4, 18, 57, 67, 98]. In the following section, different areas of REV research are explored in more detail and highlighted where they connect to the challenges of the current project.

4.1 Application-level Privacy

There exist different ways to achieve application-level privacy in REV systems. Approaches based on homomorphic encryption [25, 65] or blind signatures schemes [9, 44] have been investigated and applied in other work. Another common alternative is through the use of a Mixnet, a popular building block for verifiable and privacy preserving REV systems. Different approaches and implementations are explored in the following section.

Mixnets. The concept of a mixnet has been first introduced by David Chaum in 1981 [19] with his proposal on decryption mixes (DMN). DMNs are based on multiple encryptions of a vote. Each voter encrypts their vote once with the public key of each mixer. In reversed order, each mixer then decrypts the ballots with its private key and outputs them in permuted order, with the last mixer revealing the plaintexts. The second common form are re-encryption mixnets (RMN) first proposed by Park et al. [88] in 1993. RMNs are based on cryptographic shuffles. Initially, each voter encrypts their vote with the vote's public key. Next, the mixers shuffle the ballots, *i.e.*, they re-encrypt them using the same public key and output them in permuted order. Often, accompanied by a non-interactive zero-knowledge proof (NIZKP) attesting the correctness of the performed shuffle. Different variations of both DMNs and RMNs exist. Nowadays, RMNs and the NIZKP of correct shuffle are the *defacto* standards and very common in practical applications. They have been applied in real elections in Estonia, Norway, Switzerland and other countries [57].

Proof of Shuffle. Most RMNs require large and complex ZKPs to guarantee the re-encryption and permutation of the ballots in a privacy-preserving way. There exist two main types of shuffle proofs in literature: (1) variants based on Furukawa and Sako [45] using a proof based on permutation matrices and (2) Neff [85] using a proof based on polynomials that remain identical when permuting their roots. Variations based on Neff's proof are computationally more efficient but cannot be made non-interactive. Thus, not perfectly suited for applications with large amounts of users. In practice, the most commonly used proof variants are all based on the first type [10, 104]. One of the most well studied versions is the *Proof of a Restricted Shuffle* [104] and its implementation in the Verificatum project. Its mixnet has been used in multiple international elections of various sizes ranging from political party elections in Israel to national elections in Estonia [106]. Even though more efficient proofs exist in literature [39, 40, 61], they are not commonly used in practice due to a more complicated setup or requiring additional trust assumptions. Additionally, it is worth mentioning the Open CHVote project [52, 55] with its efforts on making Mixnet-based REV systems and the complex shuffle ZKP more approachable for non-cryptographers by providing extensive documentation and pseudo-code algorithms where necessary. Also, efforts [58] have been made in the area of automated machine-checking of all cryptographic properties in the design and implementation of the shuffle ZKPs. Due to their complexity and the resulting likelihood for mistakes, integrity checking is crucial as it is directly related to the validity and trustworthiness of a vote.

4.2 Network-level Privacy

An important promises of distributed ledger (DL) technologies is its privacy preserving aspect. First generation protocols such as Bitcoin promised but effectively failed to preserve the user's privacy as transactions could be linked back to real world users. Various attempts are made to address this problem such as anonymous broadcast or network level message shuffling which are touch upon in the following paragraphs [63]. The main idea is to mitigate the privacy attacks by decoupling the identity from a transactions.

Anonymous Communication. Zcash[1] is a second generation solution that uses cryptographic techniques such as ZKPs to avoid leaking sensitive information from transactions. Nevertheless, the approach focuses only on the DL itself and recommends the use of an anonymous communication protocol such as Tor[2] to provide network level security, *i.e.*, to prevent exposing IP-addresses [63]. Since such solutions are specifically designed for low-latency applications (*e.g.*, web-browsing, instant-messaging, etc.), they can at most provide very basic levels of privacy (see *Anonymity Trilemma* [27]). Another problem with Tor is that it is typically subject to government censorship. Therefore, it does not make sense to support a tool which ships censorship circumventing tools for voting, an activity that is not subject to censorship, as it might negatively affect the users [63]. Alternative approaches are actively researched. An example is anonymous committed broadcast that allows a set of clients to privately commit a message to a set of servers, which can then

[1] `https://z.cash/technology/zksnarks/`
[2] `https://www.torproject.org`

simultaneously open all committed messages in random order, making it impossible to create a link between the author and the message [1]. An optimal solution for anonymous publishing in permissioned DLs must aim to be directly integrated into the consensus mechanism.

Message Mixing. Another attempt trying to obscure the link between a sender and its transaction is through the use of a centralized message mixing service. Unfortunately, the centralization poses a new problem as it makes the mixing service the system's single point of failure. Also, the mixer could potentially alter transactions which are routed through it or make use of the gained information [63]. Alternative approaches are explored, such as the use of mixnets on the network-level as they are a widely accepted solution to counter traffic analysis. The problem of existing designs is that they are too computationally expensive at scale due to their need for sequential, just-in-time public-key operations. The main goal of current research is to reduce the latency in such a way that mixnets can even be used for instant messaging applications [21].

4.3 Everlasting Privacy

Both encryption as well as anonymisation techniques used to protect the voter's privacy are only as secure as the underlying cryptosystem. On top of that, encrypted votes are usually published on the Internet and, therefore, an attacker can download them and wait until the cryptosystem is broken (*e.g.*, quantum computing) or computers are sufficiently powerful to brute-force the encryption. To mitigate this threat, different approaches towards everlasting privacy, *i.e.*, the claim that privacy remains protected even if the cryptosystem is broken, exist in literature. Some ideas related to mixnets are explored below.

Mixing Commitments. In [28], one of the first publicly verifiable mixing schemes with everlasting privacy is presented. The idea is based on mixing Pedersen commitments [89]. Instead of using the message itself, a commitment is used to encode the message. Afterwards, two synchronized mixnets are run in parallel, one private, the other one public. The public mixnet operates on the commitment encoded messages and the private one on the de-commitment values which are required to open (decode) the commitments. The approach is a first step towards guaranteeing ever-lasting privacy for observers but has a drawback that allows the initial mixnet to reconstruct the messages if the underlying cryptosystem is broken. Perfectly private audit trails (PPAT) and commitment consistent encryption (CCE), introduced in [26], allow to preserve privacy even against computationally unbounded adversaries. Research [49] has shown that the mixnets based on the Terelius-Wikström shuffle ZKP [52, 104, 106] commonly used in practice are safe for use with a PPAT-CCE encryption scheme. The idea of mixing PPAT-CCE encryptions is similar to [28]. The publicly verifiable part only deals with commitments to encrypted votes and not the votes themselves. The private part deals with the encrypted commitment openings. The approach is advantageous as it uses the ElGamal encryption system

which is significantly faster than the Pailier encryption system as it can be instantiated on prime order elliptic curves.

4.4 Existing REV Systems

Many approaches and systems have been proposed throughout the years, few have been practically implemented and even less are used in practice for state- or nationwide voting. On top of that, REV is controversial in society. For some, it poses a huge security risk and threatens a country's democratic foundation whereas others consider it similar to Internet banking [35]. Since REV implementation is an effort made by governmental actors and not a privately pursued venture, the public may actively voice its concerns or even push for a moratorium such as attempted in Switzerland in 2019[3]. The following section presents a selection of efforts made to develop REV systems in literature as well as in practice.

Practical Implementations. In 2005, Estonia was the first country to introduce a REV system for all nationwide elections. The system makes use of a national wide electronic ID system which enables eligible voters to participate [47]. Nowadays, it serves as one of the most well-known examples of a country-scale REV system applied in the real world even though its privacy and verifiability properties were far from optimal initially. Throughout the years the system has been thoroughly analysed in the academic literature [62, 99] which resulted in it being much securer and more verifiable today, than when it was first introduced.

In contrast, Norway focused on a high degree of transparency and verifiability from the very beginning of the project [48, 100]. Despite the system's success, it was discontinued in 2014 but mainly due to the lack of an increase in voter turnout than for any other reason.

Switzerland already started in 2001 with initial tests on REV conducted by the *vote électronique* consortium. The project called CHVote 1.0, was developed by the Canton of Geneva [16] and was the first of its kind. Since then multiple projects have been developed in parallel all trying to fulfil the privacy and verifiability requirements published in the ordinance on electronic voting by the Federal Chancellery in 2013 [32] and in its updated version of 2018 [33]. One of these projects is the second version of CHVote focusing on achieving all necessary properties (*e.g.*, universal verifiability) such that it would be allowed for an application on national level. Despite the system's continued development and its successful tests, the canton of Geneva decided to halt all development and discontinued all existing systems due to cost reasons in 2018 [17]. Another project is the REV system developed by the Swiss Post and Scytl[4], a Spanish company providing electronic democracy services to many countries around the world. After a public intrusion test in 2019, a critical security venerability in the project's source code was discovered that allowed the creation of falsified proofs that were still verifiable without a problem [56,

[3] `https://e-voting-moratorium.ch`
[4] `https://www.scytl.com`

79]. Additionally, further problems in the documentation were found that concluded in the researches being unable to confirm the system's claimed individual verifiability property [91]. For the aforementioned reasons, the Swiss Post discontinued the trials in March 2019 and announced that they are working on a new, fully-verifiable REV system in June 2019. Thus, at the current point in time Switzerland does still not have a functioning nationwide REV system.

Research Efforts. One of the early, well-known REV systems published in literature is the work by Cramer et al. [25] from 1997. The proposal is one of the first end-to-end verifiable REV systems based on homomorphic encryption (HE). The main ideas are to tally the result of a vote in an encrypted manner and to prevent the decryption of individual ballots by distributing the private key access among multiple authorities. This work is fundamental to the design of many REV systems, among others the Provotum projects.

Another commonly known research project is Helios[5] published by Ben Adida in 2008 [3]. It is one of the first practical systems implemented from research and used in real-world elections. The system has been well studied through the years with various publications criticizing security, privacy or verifiability aspects and proposing the necessary improvements. Helios is based on client-side vote encryption and a RMN that shuffles the ballots before decryption to guarantee ballot privacy. Helios provides CaI through a challenge-or-cast scheme, RaC as voters can query their submitted ballots via their ID and UV since anyone can verify the ZKP attesting the mixer's correct execution of the shuffle.

4.5 Distributed Ledgers and REV

As shown in Section 2.2, DL technology offers many desirable proprieties for REV systems and, is therefore frequently used as a public bulletin board (PBB) to store the voters' ballots and to provide a way for public verifiability.

Research Efforts. Unfortunately, early work mostly failed to provide notable improvements over centralized architectures or was unable to scale in a practical setting. The system of [80] contains multiple single points of failure and relies on trust between a voter and different third parties (*e.g.*, voting authority, inspector). [59] proposes a system based on a private Ethereum network but again requires a trusted third party (TTP) to authenticate the voters, violating BS if the TTP should turn rogue. While the variant of the Open Vote Network (OVN) proposed in [83] offers public verifiability, the system is unfair (see definition of fairness 8) and fails to scale past 50 voters. One of the first holistic approaches to DL-enabled REV systems is provided in [15]. The authors propose various extensions to the popular Helios REV system. To improve the performance of the mixnet the original shuffle proof is replaced with Neff's verifiable secret shuffle [85]. The

[5] `https://vote.heliosvoting.org/`

architecture is adjusted to a multi authority setup using their own custom DL implementation, called Skipchain. Thereby, improving the integrity of the system and distributing the trust among a set of nodes instead of relying on a single authority. Furthermore, the global public/private key pair is replaced by a distributed key generation algorithm. An extensive overview of the current state of research on PBBs and the different commercial REV systems implementing a DL-based PBB is provided in this survey paper [102].

Provotum. The Provotum project is a proof-of-concept developed at the University of Zurich. It defines a fully decentralized REV system by distributing trust among various stakeholders and leveraging a public-permissioned DL as a PBB. The project started in 2018, with initial efforts made based on a centralized architecture using DL technology only for storage [82]. The design relied on a central server that handled the encryption of all votes and the generation of the necessary proofs. The result was a single point of failure that would compromise the privacy and trust if the entity failed. Furthermore, the system does not offer integrate verification of the generated proofs. In 2019, the first fully decentralized version leveraging client-side encryption, homomorphic tallying and a private Ethereum network was proposed. Smart contracts were used to provide auditability and public verifiability. Similar to the OVN variant, the system failed to scale past a few voters. Also, the system does not provide sufficient security levels due to technical limitations, guarantee CaI nor is it receipt-free [72]. In late 2020, a third and further refined version of the project was completed. The underlying DL technology has been replaced with Substrate[6] to remedy technical limitations imposing the inferior security levels as well as to improve the scalability. Additionally, the new version ensures receipt-freeness through the use of a ballot randomizer. Nevertheless, the system still does not provide CaI, is limited to yes/no votes and runs into trouble if too many ballots are cast at the same time [66].

[6] `https://substrate.dev`

Chapter 5

System Design

This chapter provides a high-level overview of the system design, introduces the various stakeholders and its roles and responsibilities, as well as the Provotum voting protocol implemented in Chapter 6.

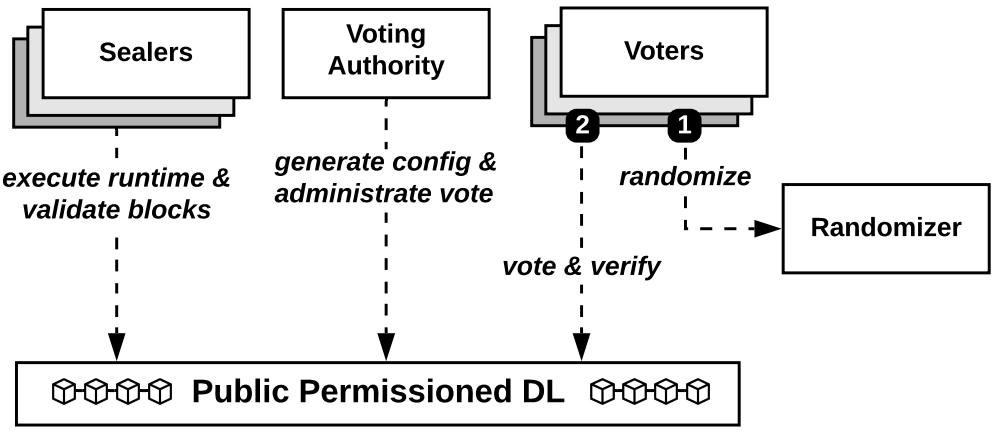

Figure 5.1: Provotum Stakeholders

5.1 Stakeholders

In the Provotum REV scheme various stakeholders exist, as illustrated in Figure 5.1, each with different roles and responsibilities which are introduced in the following section. Commonly, the scheme consists of a voting authority (VA), a randomizer, an identity provider (IDP), N sealers and M voters and a public bulletin board (PBB). The current design relies on N/N sealers being present during both the key generation as well as the cooperative decryption.

Node

A set of nodes forms the public-permissioned DL which acts as the REV vote's PBB. Using its PoA consensus mechanism, the validator nodes alternately author and validate new blocks. The functionality of the voting protocol (*e.g.*, creating a new vote, casting a ballot, etc.) is directly implemented into the DL and can be interact with via an API. Moreover, everything stored on the PBB is public and can be read by anyone, allowing for public verification of any computation performed by the PBB. Validator nodes are operated by the sealers, though other semi-trusted parties could be invited to help govern the network.

Sealers

A sealer is an entity that runs an validator node and, thereby, participates in the PoA consensus mechanism. Additionally, sealers participate in the distributed generation of the vote's public key as well as in decryption of the shuffled votes [72]. In theory, each polling station could potentially act as a sealer. In practice, it is more likely that larger municipalities or districts would run a group of sealers.

Voting Authority

The voting authority (VA) is the administrator of the vote. Its responsibility is to call the DL's API to create a vote, combine the public key shares to form the vote's public key, change the phase of a vote and to combine the decrypted shares to form the plaintext votes. In theory, the role of the VA can be handled by governmental institutions on different levels (*e.g.*, the municipality, the canton, or the national government) [72]. In practice, it is unlikely that each municipality will have the required infrastructure to securely run a VA. It is more likely, that the VA would be run in a secure data center of a canton or the national government.

Voters

Voters are people eligible to participate in the voting process. They interact directly with the DL when wanting to cast their ballot, removing the need for a trusted third party (which is common in other REV systems) and, thereby, removing a potential single point of failure in the vote casting process [72].

Randomizer

The randomizer is an entity introduced as a part of Provotum 3.0 [66] to make the ballot casting process receipt-free. The concept is based on the ideas proposed in Hirt's paper on achieving receipt-freeness in a homomorphic REV system [65]. The randomizer's task

is to blind the voter's ballot by re-encrypting it which uses a random value, only known to the randomizer. Therefore, by integrating the randomizer, a malicious voter can no longer prove to any third party how she voted since it is impossible for her to reproduce the cast ballot published on the PBB. Furthermore, it is important to note that the randomizer does not learn anything about the vote since re-encrypting a ballot is possible without knowledge of the plaintext contained in the ballot [66].

5.2 Voting Protocol

The voting protocol of Provotum is largely based on the ideas of Cramer et al. [25] and is roughly sketched as follows. The VA creates a vote and publishes the vote's system parameters. The N sealers jointly generate the vote's public key which is used by the voters to encrypt their votes. The ballots are then directly submitted to the PBB which initiates the shuffling process once the voting period is finished. Finally, the sealers jointly decrypted the shuffled votes revealing the plaintext results.

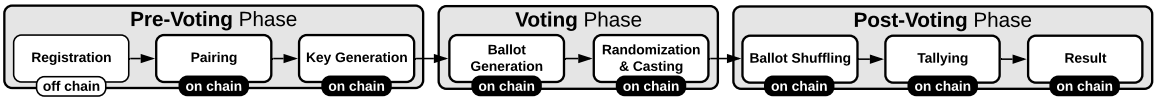

Figure 5.2: Provotum's Voting Protocol Phases

The voting protocol can be divided into three phases (Figure 5.2) and is similar to Provotum 2.0 [72] and Provotum [66]. The key differences are the absence of the *Ballot Shuffling* step in the two other protocols and the plaintext instead of HE-based tallying of the vote.

5.2.1 Identity Management

Due to the limited scope of this thesis and the extensiveness of identity management as a field of research, it is not further investigated in this work. The indentity provisioning scheme of Provotum 3.0 [66] can be used as a starting point for future research.

5.2.2 Pre-Voting Phase

In the pre-voting phase the DL network is jointly established by the authorities and the PBB is setup, the VA creates a new vote for which the sealers generate a public key. The phase consists of three main tasks: (1) registration, (2) pairing and (3) distributed key generation.

The phase starts with the **registration** step, in which each sealer generates a public/private key pair used for the identification of the sealer in the DL context. The public key is transmitted over an authenticated channel or via physical means to the VA to ensure

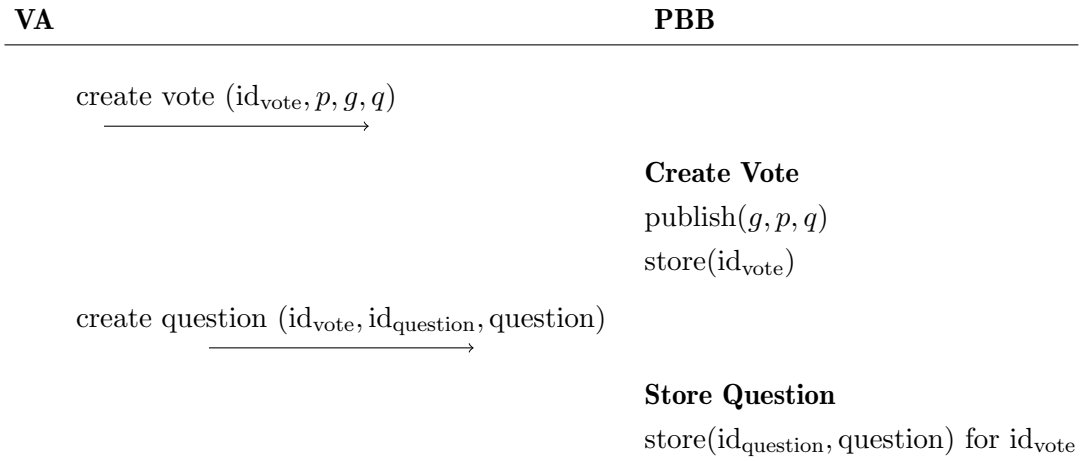

Figure 5.3: **Pre-Voting Phase** - Registration Step - Create Vote & Store Questions

the sealer cannot be impersonated by a *man-in-the-middle* adversary. Depending on the application, one or more keys are required to allow to differentiate between block production and block finalization. In the **Pairing** step, the VA generates the DL's initial configuration file, containing the sealer's public keys depicting them as validators of the network and the public key of the VA. Important to note is that the VA does not act as a validator, *i.e.*, it does not author blocks, only participate as a vote administrator. This is to ensure that the VA is not an overly powerful network peer. Finally, the VA publishes the configuration file and starts a DL node. The sealers retrieve the configuration via a public API and start a DL node themselves, together establishing the network.

Key Generation

The last step of the phase is split into three parts. First, the VA creates the vote's public parameters p, g, q and sends them together with the vote creation transaction to the PBB. The PBB validates the parameters, creates the vote and stores the associated questions (Figure 5.3).

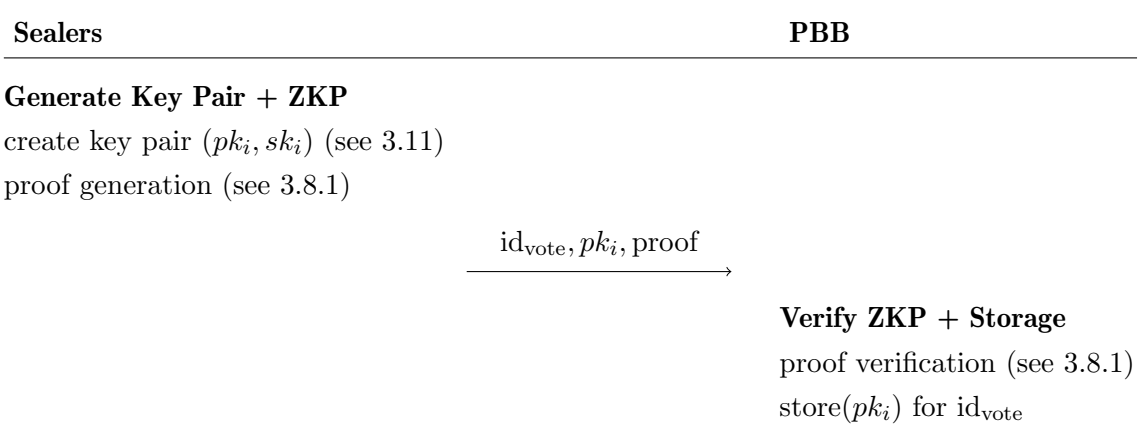

Figure 5.4: **Pre-Voting Phase** - Key Generation Step - Key Pair Generation, Proof Generation & Verification, Storage of Public Key Share

Next, each sealer $i \in N$ locally generates an ElGamal public/private key pair together with the accompanying ZKP and sends the public key share and the proof to the PBB. The PBB verifies the proof and if deemed valid, stores the public key share pk_i (Figure 5.4). Finally, once all sealers have submitted their public key shares, the VA can trigger the public key creation. The PBB combines all public key shares into the vote's public key pk_{vote} which will be later used by the voters to encrypt their votes. The public key is stored on the PBB and the vote is opened by advancing the state to the next phase (Figure 5.5).

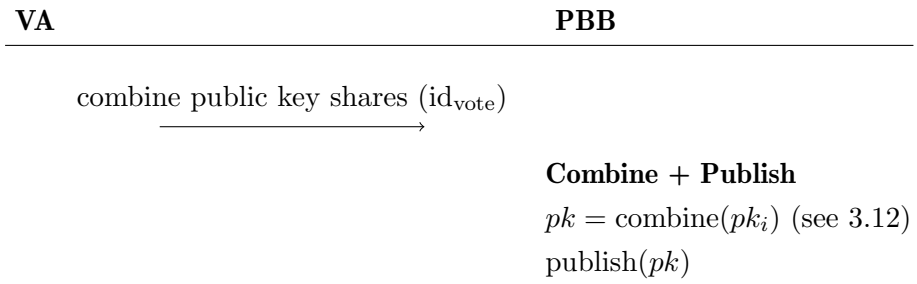

Figure 5.5: **Pre-Voting Phase** - Key Generation Step - Combine Public Key Shares to Create Vote Public Key

5.2.3 Voting Phase

In the voting phase, the voters vote, *i.e.*, answer the vote's questions, by generating their ballots, re-encrypting them using the randomizer and submitting them to the PBB. The phase is split into two steps: (1) Ballot Generation and (2) Randomization & Casting which is visualized in Figure 5.6.

Ballot Generation

In the first step, the ballot generation, the voters answer the vote's questions and generate a random ballot $c = E(v, r, pk)$ for each of their answers using the vote's public key pk for encryption.

Ballot Randomization & Casting

In the next step, each voter sends their ballots ballots to the randomizer for anonymization. The randomizer re-encrypts each ballot $c' =$ `re-encrypt(c)`, signs it with his private key and proves that the re-encryption contains the same vote v using a NIZKP. Finally, in the last part of the randomization & casting step, the voter verifies the proof and if deemed valid, submits the ballot directly to the PBB. Thus, avoiding any intermediary and, therefore, eliminating any potential single point of failure in the vote casting process. When a new ballot arrives at the PBB, the first step is to verify that the ballot is not an exact copy of an already cast ballot to avoid replay attacks. As a second step, the

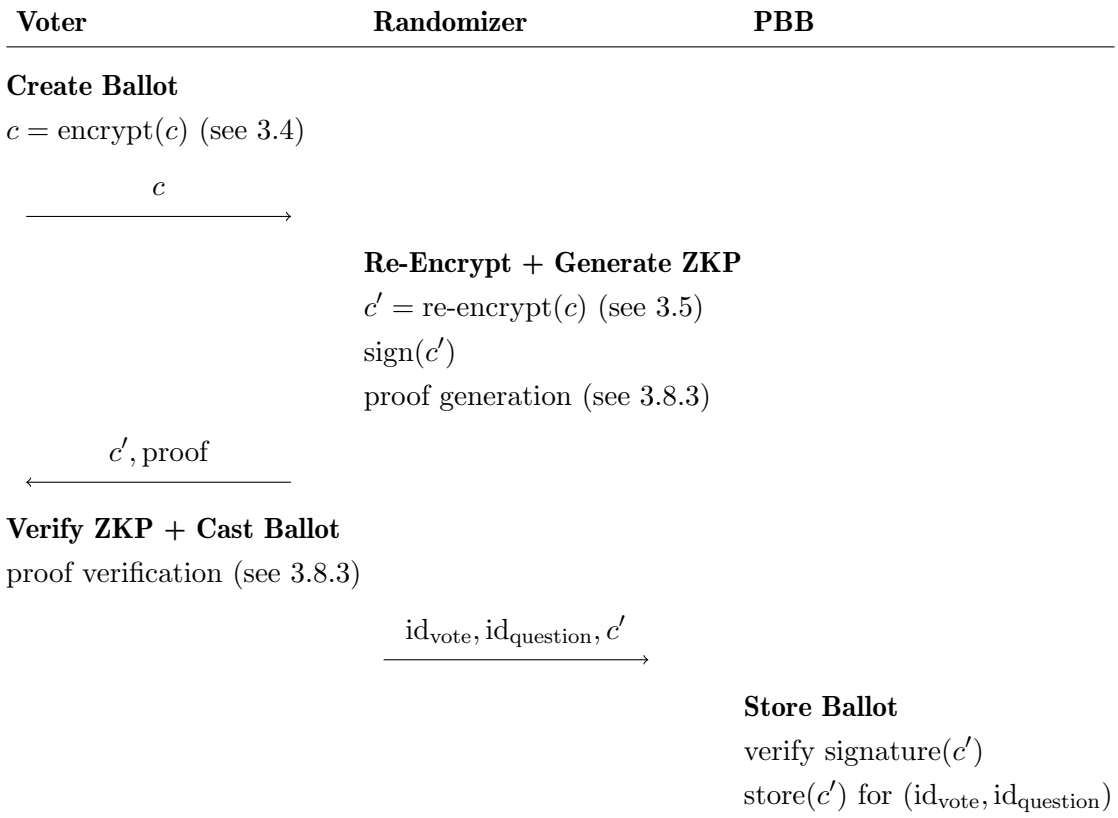

Figure 5.6: **Voting Phase** - Ballot Generation, Ballot Randomization & Casting Steps

signature of the randomizer is verified to ensure that the voter has effectively randomized the cast ballot. Without the signature verification, it would be impossible to tell if a ballot has been randomized, *i.e.*, re-encrypted, or not due to the ciphertexts being indistinguishable. Finally, the PBB stores the ballot associated with its respective vote and question identifier.

5.2.4 Post-Voting Phase

Once the voting period has ended, the VA closes the vote by advancing the state to the next phase, the post-voting phase. From now on, voting is no longer possible and new ballots are rejected by the PBB.

Ballot Shuffling

The next step, shown in Figure 5.7, is to shuffle the submitted ballots so that they can no longer be linked to their creators. For this, the sealers retrieve a copy of the ballots **c** from the PBB and initiate the process. The shuffling takes place outside of the DL context since random values, which are not supposed to be leaked, are required in both the re-encryption as well as the permutation. The downside of this is that the performed shuffles are not publicly verifiable which is crucial to establish trust in the process and

the correct functioning of the REV system. To circumvent this problem and to allow for public verifiability, the sealers prove correctness of the shuffle by generating a NIZKP that proves knowledge of the random values as part of the shuffle.

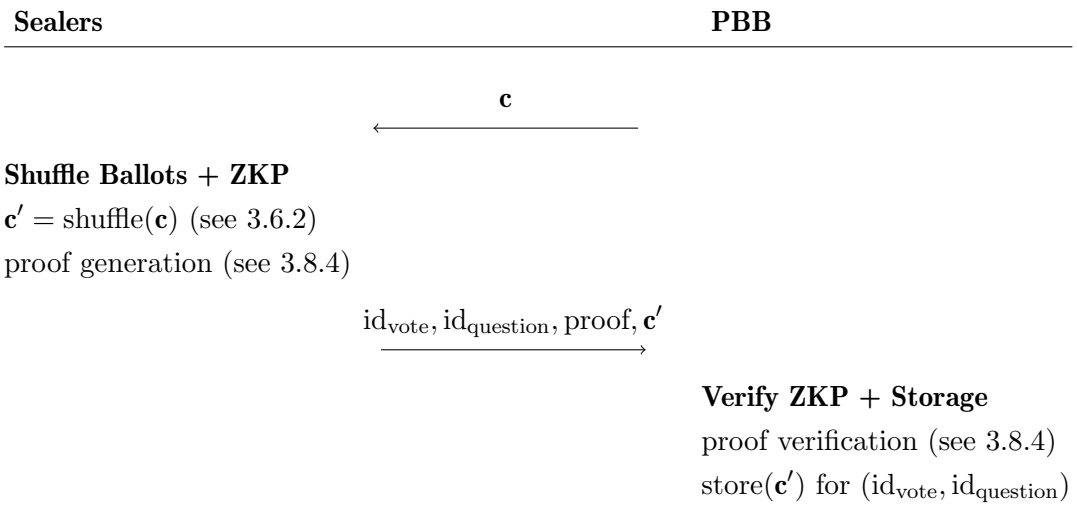

Figure 5.7: **Post-Voting Phase** - Ballot Shuffling Step - Ballot Shuffling, Proof Generation/Verification and Storage

Finally, the proof is submitted together with the shuffled ballots $\mathbf{c}'$. The PBB verifies the proof and stores the shuffled ballots, if the proof is valid. The process is repeated by a different sealer using the shuffled ballots $\mathbf{c}'$ as input producing the next generation of shuffled ballots $\mathbf{c}''$. The shuffling stops once the desired number of shuffles is reached. This number can be specified in the initial configuration and varied from a vote to vote basis. It is suggested to perform more than a single shuffle to avoid recoverability in the case of a malicious sealer peer. In case of a large number of ballots, the ballots can be split into batches such that multiple sealers can simultaneously shuffle a different batch of the same generation of ballots.

Tallying

Once all ballots have been sufficiently shuffled, the VA can initiate the distributed decryption of the vote. A visualization of this step is shown in Figure 5.8. For this, each sealer $i \in N$ partially decrypts the shuffled ballots $\mathbf{c}''$. This part occurs outside of the DL context as it involves the sealers' private key shares sk_i. To ensure that the partial decryption $\mathbf{d}_i$ has been performed correctly, the sealers create a proof showing knowledge of the private key share sk_i that belongs to the public key share pk_i, they published to the PBB during the distributed key generation (5.2.2). The partial decryption $\mathbf{d}_i$ is submitted together with the proof to the PBB. After successful proof validation, the partial decryption is stored.

It is important to note that in the current implementation all N sealers that participated in the key generation must also participate in the distributed tallying. If any sealer (purposely) refrains from submitting their partial decryption, the ballots cannot be decrypted.

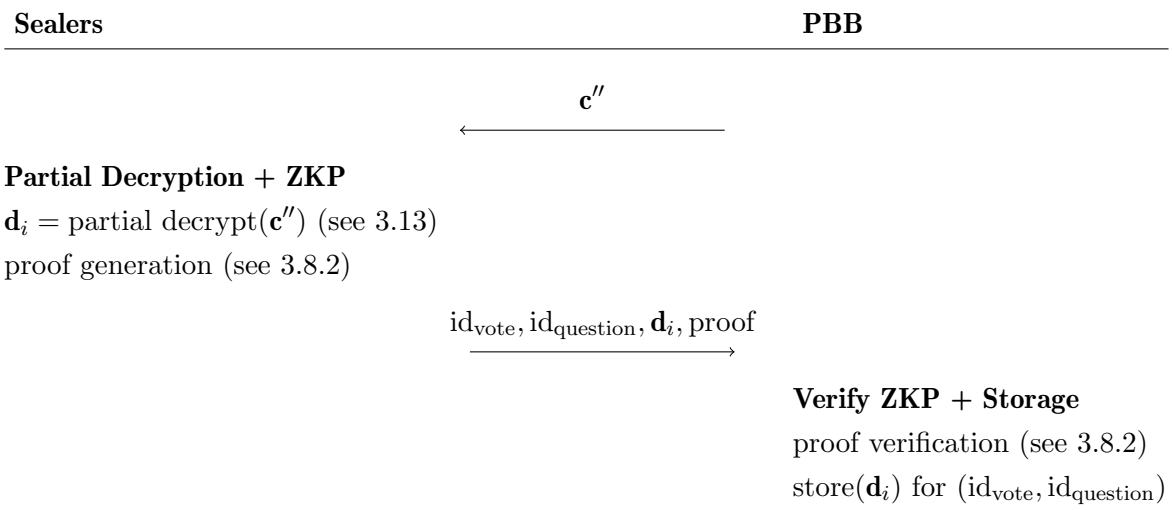

Figure 5.8: **Post-Voting Phase** - Tallying Step - Ballot Shuffling, Partial Decryption

Results

The last step of the phase is composed of the combination of the partial decryptions $\mathbf{d}_i$, the public decryption of the combined partial decryptions $\mathbf{d}$ and optionally, the decoding of the resulting plaintext votes. The step, visualized in Figure 5.9, is initiated by the VA triggering the combination of the partial decryptions $\mathbf{d}_i$. The PBB ensures that all sealers have submitted their partial decryptions and combines them ballot-wise. Next, the combined partial decryptions $\mathbf{d}$ are decrypted using the remaining component c_{i_2} of the encrypted ballots $\mathbf{c}''$, revealing the result. Depending on the message encoding used, the result needs to be decoded before the plaintext votes can be revealed.

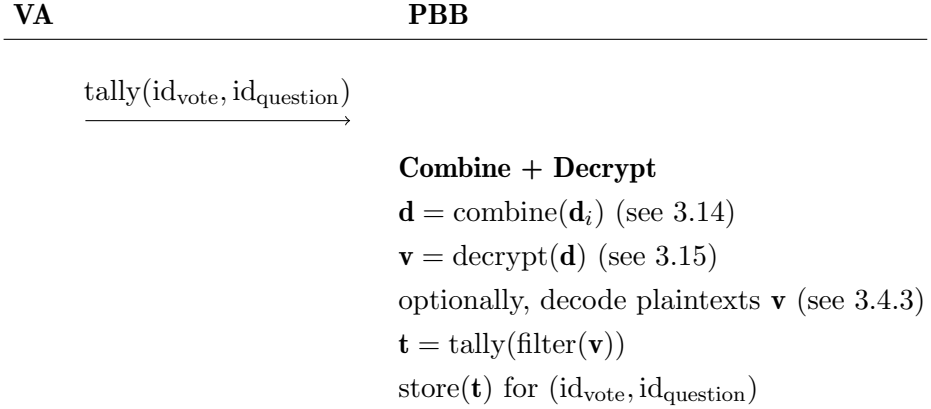

Figure 5.9: **Post-Voting Phase** - Results Step - Combine Partial Decryptions, Decrypt and Reveal Plaintexts

In the last step, the plaintext votes are filtered to ensure any invalid votes $\mathbf{v}$ or incorrect submissions are not counted. Finally, the filtered set is tallied and for each unique entry its count is denoted. Once the PBB has stored and published the final tally, the voting protocol is completed.

Chapter 6

Implementation

Alongside this thesis, a prototype has been developed implementing the system architecture proposed in Chapter 5. The prototype is structured into five individual packages as shown in Figure 6.1. The interactions are highlighted in red and the dependencies in blue. The source code for the individual packages is published in the `provotum-mixnet` repository which is part of the Provotum GitHub organization[1].

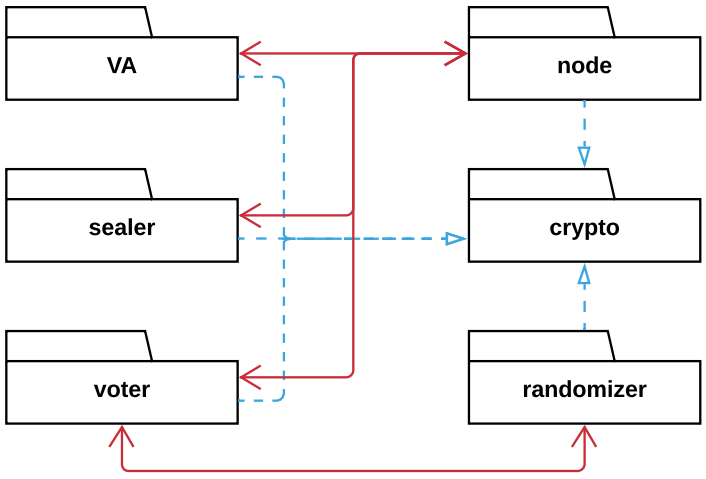

Figure 6.1: Provotum Prototype Packages

6.1 Packages

The `node` package is the main component of the system. It implements the public bulletin board (PBB) and the main parts of the voting protocol using Substrate[2], a modular framework developed by Parity Technologies[3] that allows to create purpose-built distributed

[1] `https://github.com/provotum`
[2] `https://substrate.dev`
[3] `https://www.parity.io`

ledgers (DL). The `node` package is composed of various components, many provided by the maintainers of Substrate, abstracting the complexity necessary for core functionalities such as networking, communication and data storage. Thus, allowing the user to focus on developing the runtime logic. The voting protocol and mixnet implementation are contained in its own Substrate package, called *pallet-mixnet*. Its main entry point is shown in Figure 6.2. The main idea behind this is to avoid tightly coupling the implementation to rest of the runtime logic, such that the pallet could potentially be replaced (*e.g.*, by a homomorphic encryption pallet) or used in a different Substrate project. Substrate, as well as all other packages of this prototype, are developed using Rust[4], a modern, low-level, type- and memory-safe language heavily focused on performance. To allow for straightforward testing, simplified project exploration and with reproducibility in mind all packages are containerized using Docker[5]. The existing images are available from the Provotum GitHub organization. Additionally, a docker-compose[6] script is provided that allows to start a pre-configured test setup with the ease of a single command.

```
1  // Imports dependencies from Substrate
2  use frame_support::{decl_module, decl_event, decl_storage, decl_error};
3
4  // Imports functionality defined in this pallet
5  use crate::helpers::phase::set_phase;
6
7  // Imports functionality defined in the crypto library
8  use crypto::proofs::shuffle::ShuffleProof;
9
10 // Runtime Configuration: Defines the traits that this configuration
   ↪  depends on and that this pallet needs to implement
11 pub trait Trait: pallet_timestamp::Trait + frame_system::Trait +
   ↪  frame_system::offchain::CreateSignedTransaction<Call<Self>> { ... }
12
13 // Storage Declaration. Defines the storage type (simple value or
   ↪  key/value map) and the access level (private/public)
14 decl_storage! { ... }
15
16 // Event Definition. Defines the different events including their
   ↪  contained values and/or messages
17 decl_event! { ... }
18
19 // Error Definition. Defines the different errors that can occur
20 decl_error! { ... }
21
22 // Extrinsics (API Definition). Defines the publicly callable functions
   ↪  that can interact with the runtime and read/modify the storage. Also,
   ↪  it defines the offchain functionality
23 decl_module! { ... }
```

Figure 6.2: `node/pallets/mixnet/src/lib.rs`: the main entry point of the mixnet pallet.

Another important package of this prototype is the `crypto` package. It implements the algorithms and zero-knowledge proofs (ZKP) required by the voting protocol. The com-

[4] `https://www.rust-lang.org`
[5] `https://www.docker.com`
[6] `https://docs.docker.com/compose`

ponent is packaged as a library such that it can be included in the `node` package and used by any other package (*e.g.*, randomizer, VA, sealer, voter) wanting to make used of its functionality. It abstracts away the complexity of the algorithms and proofs implementation by exposing a public API that can be interacted with. A user does not need to know about or understand implementation details to be able to use the library. Also, by extracting the functionality to a separate package, it was possible to test it extensively in isolation. For a Rust package, called *crate*, to be used with Substrate, it must be compiled for the Web Assembly[7] target `wasm32-unknown-unknown` and be `no-std` compatible which means that it cannot make use of the standard library (*e.g.*, no I/O operations, no dynamically sized values such as strings or collections). The package has been specifically designed for this use case. Nevertheless, it can also be used in a `std` compatible and if compiled accordingly using the respective feature flag, additional features such as random number generation are available.

The third independent package is the `randomizer`. It is a simple web-based service that implements the ballot re-encryption and designated verifier ZKP generation using the algorithms provided in the `crypto` package. The component is dockerized stateless service that can be scaled according to the needs of the application.

```
provotum-cli 1.0
Moritz Eck <moritz.eck@gmail.com>
The Provotum CLI to impersonate voters, the voting-authority and sealers

USAGE:
    provotum-cli <SUBCOMMAND>

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

SUBCOMMANDS:
    help      Prints this message or the help of the given subcommand(s)
    sealer    A subcommand for controlling the Sealer
    va        A subcommand for controlling the Voting Authority
    voter     A subcommand for controlling the Voter
```

Figure 6.3: The default message shown by the `provotum-cli`. For each component, a different subcommand provides the built-in operations.

The remaining components of the prototype are the `VA`, the `sealer` and the `voter` which are test clients developed to simulate different vote setups and scenarios. Each component personifies a stakeholder, as the names already suggest. The components are bundled inside the `client` package into a single command line interface called `provotum-cli`. The default message of the `provotum-cli` can be seen in Figure 6.3. The `client` uses the functionality provided by the `crypto` library, the Rust Substrate client `substrate-subxt`[8] to interact with the `node` package and `surf`[9], a Rust HTTP client, to interact with the `randomizer`.

[7] `https://webassembly.org`
[8] `https://github.com/paritytech/substrate-subxt`
[9] `https://github.com/http-rs/surf`

6.2 Documentation

All packages develop as part of this prototype are extensively documented with the goal to make it as simple as possible for other developers to understand what currently exists, how it works and to get an idea of how it can be extended. In the following, two examples (Figure 6.4 and Figure 6.5) from the `crypto` package are shown for illustration purposes.

```
1  /// Returns an ElGamal Encryption of a message. The message
   ↪  encoded such that additive homomorphic operations are
   ↪  possible i.e. g^m_1 * g^m_2 = g^(m_1 + m_2)
2  /// - (a, b) = (g^r, pk.h^r * g^m)
3  ///
4  /// ## Arguments
5  ///
6  /// * `m`  - The message (BigUint)
7  /// * `r`  - The random number used to encrypt_encode the
   ↪  vote
8  /// * `pk` - The public key used to encrypt_encode the vote
9  pub fn encrypt_encode(m: &BigUint, r: &BigUint, pk:
   ↪  &PublicKey) -> Cipher { ... }
```

Figure 6.4: The figure shows the function header and documentation for `encrypt_encode` which encodes and encrypts a message/vote such that it is possible to perform additive homomorphic operations on the ciphertext. The code is found in the following file: `crypto/src/encryption.rs`

Whenever an implementation is based on existing work, it is referenced accordingly. For example, the implementation of the shuffle proof and its dependent functionality is inspired by Haenni et al. [52] and their work on the CHVote Specification[10]. Figure 6.5 shows some functionality that exists in a similar manner (*i.e.*, not vector-based) in the CHVote specification and has been adapted for the use in this work.

6.3 Technical Limitations

Due to the limited scope of this thesis and the extensiveness of identity management as a field of research, the prototype implementation does not take eligibility verification and voter identity management into consideration. During the ballot casting, the voter's identity is not verified. Hence, a voter can cast multiple ballots. This can be easily fixed, as the necessary pointers are provided in the source code. Additionally, the public/private key pairs of the authorities as well as the randomizer are directly integrated into the configuration of the prototype. In a real-world setup, the key pairs would have to be generated in a secure (potentially air gaped) environment before each vote and inserted from the outside once the DL network is running. Two different use cases are pre-configured. A `dev/testing` setup with a single authority able to perform all operations and a more realistic setup, the `local` configuration composed of a voting authority (Alice) and two

[10] `https://eprint.iacr.org/2017/325.pdf`

```rust
/// Similar to GetVotes Algorithm 8.53 (CHVoteSpec 3.2)
///
/// Computes the decrypted plaintext vote m by deducting the combined partial decryptions decrypted_a = a^sk = (g^r)^sk) = g^(sk*r) from the left-hand side b of the encryption e = (a, b) = (g^r, pk^r * m).
///
/// b = pk^r * m = (g^sk)^r * m = g^(sk*r) * m
/// m = b / g^(sk*r) = b * (g^(sk*r))^(-1)
/// m = b * inverse_mod(g^(sk*r)) mod p
/// Returns plaintext vote: m | encoded(m)
///
/// ## Arguments
///
/// * `b` - The component b of an ElGamal Encryption (a: BigUint, b: BigUint)
/// * `decrypted_a` - The decrypted component a of an ElGamal Encryption
/// * `p` - The group modulus p (BigUint)
pub fn partial_decrypt_b(b: &BigUint, decrypted_a: &BigUint, p: &BigUint) -> BigUint {
    let s_1 = decrypted_a.invmod(p).expect("cannot compute mod_inverse!");

    // b = m * pk^r -> m = b * ((g^sk)^r)^-1
    b.modmul(&s_1, p)
}
```

Figure 6.5: The figure shows the function documentation and implementation for `partial_decrypt_b` which decrypts the component b of an ElGamal encryption $\mathbf{e}$ = (a,b) given a partially decrypted $\mathbf{e}$. The code is found in the following file: `crypto/src/encryption.rs`

sealers (Bob and Charlie). These technical shortcomings have been addressed in Section 8.2 providing starting points for further research and future theses.

Chapter 7

Discussion and Evaluation

In this chapter, the proposed voting protocol and system architecture are evaluated regarding privacy, verifiability, scalability as well as different practical properties. The evaluation is separated into four different parts. In the first two sections, an analysis of the privacy and verifiablity properties is performed. Next, the achieved practical properties are investigated. In the last section, the prototype is evaluated in terms of scalability and real-world scenario applicability.

7.1 Privacy

The following section evaluates and discusses the privacy properties in relation to the proposed voting protocol. In Table 7.1 an overview of how the privacy properties of the Provotum REV system evolved across projects is presented. All projects apart from the second major version of Provotum provide ballot secrecy (BS) and receipt-freeness (RF).

	BS	**E-BS**	**RF**	**CR**
Provotum 2.0 [72]	✓	✗	✗	✗
Provotum 3.0 (HE) [66]	✓	✗	✓	✗
Provotum 3.0 (Mixnet)	✓	✗	✓	✗

Table 7.1: Visualization of the privacy properties across all three Provotum projects.

7.1.1 Ballot Secrecy

In the proposed voting protocol, ballot secrecy is guaranteed as the ballots stored on the public bulletin board (PBB) in the REV system are encrypted using the public key. No single entity is able to decrypt individual ballots as the public key pk has been jointly generated by the sealers using their shares pk_i (see Section 3.7.1). Even if a majority of sealers were to turn rogue and collude on decrypting the ballots, it would remain impossible as long as a single sealer remains truthful. Also, it is important to note that

the randomizer does not learn anything about a vote, as it simply computes an encryption of zero which is homomorphically added to the ballot, making it indistinguishable.

Since the security of the voting protocol is based on the security of the ElGamal cryptosystem and, therefore, on the DDH assumption 13, everlasting privacy cannot be guaranteed. It is possible that at some point in the future, the DDH assumption will no longer hold due to the availability of a sufficiently powerful quantum computer. Thus, leading to far more wide ranging problems than just the insecurity of this voting protocol as all systems based on public-key cryptography will no longer be secure. At the end of 2020, between 50 and 100 qubits were the best publicly known results of existing quantum computers [7]. In comparison, the current assumptions are that to factor an ElGamal private key of 2048 bit size anything between twice the number, *i.e.*, 4096 qubits, and a billion qubits would be required [92].

7.1.2 Receipt-Freeness

To show that the proposed voting protocol is receipt-free, two important aspects are considered. One, no information about the vote is leaked. As the voter encrypts the vote using a random value r_1 and the randomizer re-encrypts the voter's ballot using another random value r_2, it is impossible to reproduce the result as neither one of the two knows both random values. Also, if the voter attempted to prove to a vote buyer that she cast her vote in a specific way, it would be impossible for her to do so as the randomizer always uses a different random value r_2' for a different randomization. Only if the randomizer and the voter were to collude, together they would be to prove to a vote buyer that the voter cast a certain ballot. The second aspect to consider is that the NIZKP used by the randomizer to prove to the voter that the randomized ballot is a valid re-encryption, is a designated-verifier proof. This is important as the proof is only of use to the voter and cannot be transferred to a third-party. Thus, unable to convince a vote buyer.

Depending on the thread model and risk assessment a different kind of randomizer can be employed. In any case, an unauthenticated, untappable channel between the voter and randomizer is required. In high coercion environments, a randomizer based on a hardware token, as known from Internet Banking, could be used whereas in other scenarios it might be feasible to deploy a web service. The later version is used in the prototype's implementation.

7.1.3 Coercion-Resistance

Although the protocol achieves BS and RF, it is not coercion resistant (CR). A voter can be forced to give up her voter credentials or to abstain, if a coercer is physically present at the same location. Also, the coercer can force a voter to vote in a specific way and ensure the vote casting by observing the process in person. Other countries with established REV systems (*e.g.*, Estonia), circumvent this problem by allowing voters to cast their ballots multiple times, each submission overriding the previous one, and to override their electronically submitted ballots in person on the voting day [99]. This allows a coerced

voter to invalidate the previously forced ballot. Unfortunately, the Swiss law [29] explicitly forbids the repeated ballot casting *i.e.*, each eligible voter is only allowed to cast their ballot once. Thus, making it impossible to guarantee CR in a REV setting in Switzerland without simultaneously providing a physical voting alternative.

7.2 Verifiability

The following section evaluates and discusses the verifiability properties in relation to the proposed voting protocol. In Table 7.2 an overview of how the verifiability properties of the Provotum REV system evolved across projects is presented. All projects starting from the second major version of Provotum provide individual (IV) and universal verifiability (UV) as well as the properties recorded-as-cast (RaC) and counted-as-recorded (CaR).

	IV	**UV**	**E2E-V**	**CaI**	**RaC**	**CaR**
Provotum 2.0 [72]	✓	✓	✗	✗	✓	✓
Provotum 3.0 (HE) [66]	✓	✓	✗	✗	✓	✓
Provotum 3.0 (Mixnet)	✓	✓	✗	✗	✓	✓

Table 7.2: Visualization of the verifiability properties across all three Provotum projects.

7.2.1 Cast-as-Intended Verifiability

In the proposed voting protocol, CaI verifiability is not guaranteed for the layperson as she does not have a way to audit the content of a ballot. In the current design, the REV system encrypts the vote, sends the ballot to the randomizer for re-encryption, verifies the returned NIZKP and publishes both to the PBB. To audit a ballot stored on the PBB, a voter needs to record the random value used during the initial encryption to recompute the pre-randomization ballot. This process is complex and requires an understanding of the cryptographic primitives used which is an infeasible assumption to make in a system designed for the masses. The fact that REV systems allow voting from uncontrolled and insecure devices reinforces the need for a more user-friendly audit possibility. The shortcoming has been addressed in Section 8.2 providing a starting point for future theses and is already being addressed by a master thesis running in parallel to this work.

7.2.2 Individual & Recorded-as-Cast Verifiability

After casting the randomized ballot and the re-encryption NIZKP the PBB returns a transaction hash confirming its inclusion. As soon as all distributed ledger (DL) network peers have agreed to include the block which contains the voter's transaction hash, the voter can verify that its submitted ballot has been included by checking the transaction log or querying the storage. By comparing the ballot from the PBB with the ballot stored on the voter's device, the voter can verify that the ballot has been recorded as cast. This also fulfils the requirements defined for IV (5).

7.2.3 Counted-as-Recorded

The usage of a DL as a PBB allows anyone to verify the correctness of the vote's result. All computations and proofs are stored by the PBB and once consensus is reached among the DL network peers publicly available for verification. Proof verification and computations part of automated PBB functionalities (*e.g.*, combining public key or decrypted shares) can be recomputed manually or using a verification software. A verifier would immediately notice any manipulation attempts as, for example, replacing a ballot during the shuffling process would stand out since the accompanying proof would fail during the verification. In the following more examples are given.

Invalid Ballots. Should a ballot be invalid, as the vote contained is empty or garbage, would this be recognized after the decryption and before the tallying of the plaintexts. All invalid votes can then simply be discarded and excluded from the result. As long as this is communicated by the voting authority when publishing the results, the process and results remain verifiable.

Order of Ballots. The order of the ballots and the decrypted shares in the respective ZKPs is inherently given as the PBB verifies the proofs by recomputing all necessary values in the same order as the input values are stored. Therefore, the proof verification will always fail should a sealer use ballots or decrypted shares in a different order in the proof generation than the PBB will use during verification.

7.2.4 Summary

For a REV system to claim to be E2E verifiable, it must fulfil all of the previously mentioned properties: CaI, RaC, CaR. Since the voting protocol in its current design does not achieve CaI for the layman and the masses, E2E verifiability cannot be guaranteed. On the other hand, universal verifiability (UV) is ensured as both RaC and CaR are given.

7.3 Practical Properties

The following section evaluates and discusses the properties relevant for a practical deployment of the proposed voting protocol. In Table 7.3 an overview of how the practical properties of the Provotum REV system evolved across projects is presented. All projects starting from the second major version of Provotum provide fairness and transparency for basic yes/no votes.

	Fairness	Accountability	Robustness	Transparency	1+	Yes/No	Elections
Provotum 2.0 [72]	✓	✗	✗	✓	✗	✓	✗
Provotum 3.0 (HE) [66]	✓	✗	✓	✓	✓	✓	✗
Provotum 3.0 (Mixnet)	✓	✗	✓	✓	✓	✓	✓

Table 7.3: Visualization of the practical properties across all three Provotum projects.

7.3.1 Fairness

Fairness, as defined in Definition 8, is guaranteed in the voting protocol through the distributed key generation process and ensured as long as at least a single sealer remains honest. Thus, it remains impossible to decrypt the ballots prematurely. Only once the voting period has ended are the sealers officially tasked with shuffling and tallying the vote.

7.3.2 Accountability

In the proposed protocol design, misbehaving parties can be identified. A sealer no longer taking part in the DL consensus mechanism would not go unnoticed as well as if it tried to alter ballots during a shuffle or not submitted its decrypted shares for a particular vote and question.

Due to the fact that n/n sealers taking part in the distributed key generation need to take part in the distributed tallying, a malicious sealer can only be identified but not excluded from the process without aborting the vote. It is essential for future designs that misbehaving parties can be excluded without affecting the outcome of a vote. The shortcoming has been addressed in Section 8.2 providing a starting point for future theses.

7.3.3 Robustness

As the proposed REV system makes use of a DL as PBB, robustness as defined in Definition 10, is inherently provided. The DL is operated as a P2P network with each peer storing the complete state itself, avoiding the loss of state information due to a single point of failure. Additionally, the network is maintained and state changes are agreed upon using a majority vote among the authorities. Even if some network participants $t < \frac{N}{2}$ were to go offline or fail to take part in the block authoring and validation process, the REV system would continue to function normally.

7.3.4 Votes and Elections

Up until Provotum 3.0 only simple binary votes were supported. By replacing the HE-based tallying component of the Provotum REV system with a Mixnet-based approach, it removed the restrictions on votes being either zero or one. This allows for limited votes

where voters select k out of N candidates or multi-way elections where voters select a single candidate out of N possibilities.

In some countries, such as Switzerland, write-ins, *i.e.*, votes containing a handwritten name of a candidate, are allowed during the voting process as part of suggesting additional candidates. This is currently not implemented in the prototype but technically possible. The CHVote specification [52] provides a good starting point for further research.

7.4 Scalability

For a REV system to be applicable in practice, it not only needs to achieve the posed security, privacy and verifiability requirements but also scale with the number of users. To show that the proposed voting protocol and prototype implementation can be used in a real world vote, we first provide a theoretical runtime analysis by counting the of modular exponentiations (`modExp`) and comparing them with the previous projects. In the second part, we provide a quantitative performance evaluation using different benchmarks.

7.4.1 Theoretical Runtime Performance

The performance of cryptographic algorithms is often compared in terms of `modExp` as those are the most computationally intensive operations when using sufficiently large (*i.e.*, 2048 bits and more) key sizes for security reasons [53, 54]. In Table 7.4, the computational cost of casting a single ballot in Provotum 2.0 and 3.0 are put into comparison with this work. While Provotum 2.0 required 17 `modExp` per ballot, Provotum 3.0 required an additional 26 `modExp` due to the additional operations necessary to guarantee RF in a HE-based REV system, totalling in 43 `modExp` per ballot. This work only requires an additional 8 `modExp` over Provotum 2.0 and results in a reduction of 18 `modExp` when compared to Provotum 3.0. There are different optimizations and changes explained in the following resulting in the reduced computational overhead.

	Provotum 2.0		**Provotum 3.0**			**This Work**			
Step	Voter	DL	Voter	DL	Randomizer	Voter	DL	Randomizer	Sealer
Ballot Encryption	3		3			2			
Ballot Validity ZKP Generation	6		6						
Re-Encryption ZKP Generation					2			2	
Re-Encryption ZKP Verification			2			2			
Ballot Validity ZKP Randomization			6		16				
Ballot Validity ZKP Verification		8		8					
Ballot Shuffling									2
Shuffle ZKP Generation									8
Shuffle ZKP Verification							9		
Total (Entity)	9	8	17	8	18	4	9	2	10
Total	**17**		**43**			**25**			

Table 7.4: Comparison of cryptographic operations across the existing Provotum projects in terms of `modExp`.

Ballot Validity ZKP. Compared to HE-based REV systems, in a Mixnet-based approach all ballots are decrypted before the tallying step and, therefore, their validity can be checked. This removes the requirement for the ballot validity proof, *i.e.*, the zero-knowledge membership proof showing that the vote contained in the ballot is part of a set of allowed votes ($v \in 0, 1$). In terms of runtime performance, this reduces the number of `modExp` required by 14 respectively 36 compared to Provotum 2.0 and 3.0. Additionally, by removing the ballot validity ZKP as used in Provotum 3.0, the voting protocol no longer relies on a weak Fiat-Shamir transformation and, therefore, the assumption that both the voter and the randomizer remain honest. The weak Fiat-Shamir transformation is flawed as it allows a malicious prover to generate a proof that verifies correctly even if the asserted value does not lie in the set of allowed votes [12].

Ballot Shuffling. To render the connection between a voter and its votes unrecoverable even though the ballots are decrypted before the tallying, the ballot shuffling is required. This process is composed of offline shuffling, offline proof generation and online proof verification by the PBB. This process does not exist in the previous Provotum projects and occurs an additional computation overhead. The shuffling operation requires 2 `modExp` per ballot while the proof generation requires a constant 5 `modExp` and an additional 8 `modExp` per ballot. The online proof verification requires a constant 11 `modExp` and an additional 9 `modExp` per ballot. Since shuffling only makes sense with more than a couple of ballots and due to the constant parts of the proofs, one could argue that the larger the shuffle size the better. The assumption is further investigated in the quantitative evaluation.

7.4.2 Benchmark Results

Theoretical runtime estimations are useful to identify the magnitude of algorithmic performances but do not take real world overhead and implementation differences into account. To estimate the practical scalability of the prototype REV system, we benchmark the performance of the algorithms in a real-world scenario using 1'000'000 ballots as the target size of the vote. The benchmarks are performed on a Ubuntu 20.04 server using a 12-core AMD Ryzen 3900X CPU, base clock of 3.8GHz, and 64GB RAM. The results were established by running each algorithm 10 times. In the following, the benchmarks are group according to the vote phase they belong to.

Pre-Voting Phase

The pre-voting phase, composed of pairing, distributed key generation and vote setup, does not contain any operation that needs to scale past a few number of participants. Thus, no benchmarks have been performed for this phase.

Voting Phase

In the voting phase, the ballot randomization as well as the vote casting need to scale. Both are benchmarked and the results are discussed in the following. On the other hand,

the ballot generation does not need to scale as each voter performs this operation on its device. But since the vote encoding strategy impacts the decryption performance, a benchmark of the encoding strategies has been performed.

Vote Encoding. As the vote encoding used in this work (introduced in Section 3.4.3) is different compared to the previous projects, the two strategies are benchmarked against each other. In Table 7.5, the performance differences are visualized. In this work, the encryption is about 10x more time-consuming as for each vote before encrypting it the quadratic residue check is performed. Since this operation only occurs once per ballot and on the voter's devices, the performance degradation is negligible. On the other hand, the performance of the decryption is constant and between 2x and 50x more efficient, depending on the message input space, as the result of the decryption does not have to be decoded to reveal the plaintext. In the former prototype implementations, the encoded result had to be be brute forced to retrieve the corresponding plaintext. But as only a single ciphertext, *i.e.*, the homomorphically summed result, had to be decrypted and decoded this was feasible. If we were to apply the same encoding strategy in this work, where (1) votes are not restricted to zero and one and (2) each ballots needs to be decrypted before the result can be tallied, the approach would not scale past a few thousand ballots depending on the size of the vote space.

	Encryption	**Decryption**		
		$m \in [0, 10]$	$m \in [0, 100]$	$m \in [0, 1000]$
Encoding 1: g^m (3.4.3)	0.66 *ms*	4.21 *ms*	17.09 *ms*	144.93 *ms*
Encoding 2: $m^q \mod p \equiv 1$ (3.4.3)	4.37 *ms*	2.87 *ms*		

Table 7.5: The performance difference of the two message encoding strategies (see Section 3.4.3) for p of size 2048 *bits* and $q = \frac{p-1}{2}$.

Randomizer. Since re-encrypting a ballot as well as generating a designated verifier ZKP are stateless operations, they are no performance bottleneck even if the operations itself were not efficient. Additional load can simply be handled by scaling up more randomizer services.

Ballot Casting. Finally, the performance of the ballot casting is relevant as it is crucial for the number of incoming ballots the DL network can handle at a time. The time required to process a ballot is only 0.018 *ms* as no processing apart from the ballot duplication check and the randomizer's signature verification is required. Thus, compared to the previous prototype implementations, this work is not limited by the ballot casting throughput. The second factor that needs to be considered is the number of transactions a DL node can handle per time interval. The ballot casting is benchmarked for the default block time of 6 seconds. The performance of the network, composed of two sealers and a voting authority, was able to handle $\approx$ 2100 transactions per block. This equates to 350 transactions per second or 1.25 million per hour.

Post-Voting Phase

In the post-voting phase, the shuffle ZKP verification and the partial decryption ZKP verification by the PBB need to scale as the shuffling, tallying and publishing of the results should not take longer than in a regular/postal voting scenario. Therefore, both operations are benchmarked and the results are discussed in the following. Additionally, the performance of the ballot shuffling in comparison to the chosen group modulus p is investigated to highlight the impact on performance of increasing the security level.

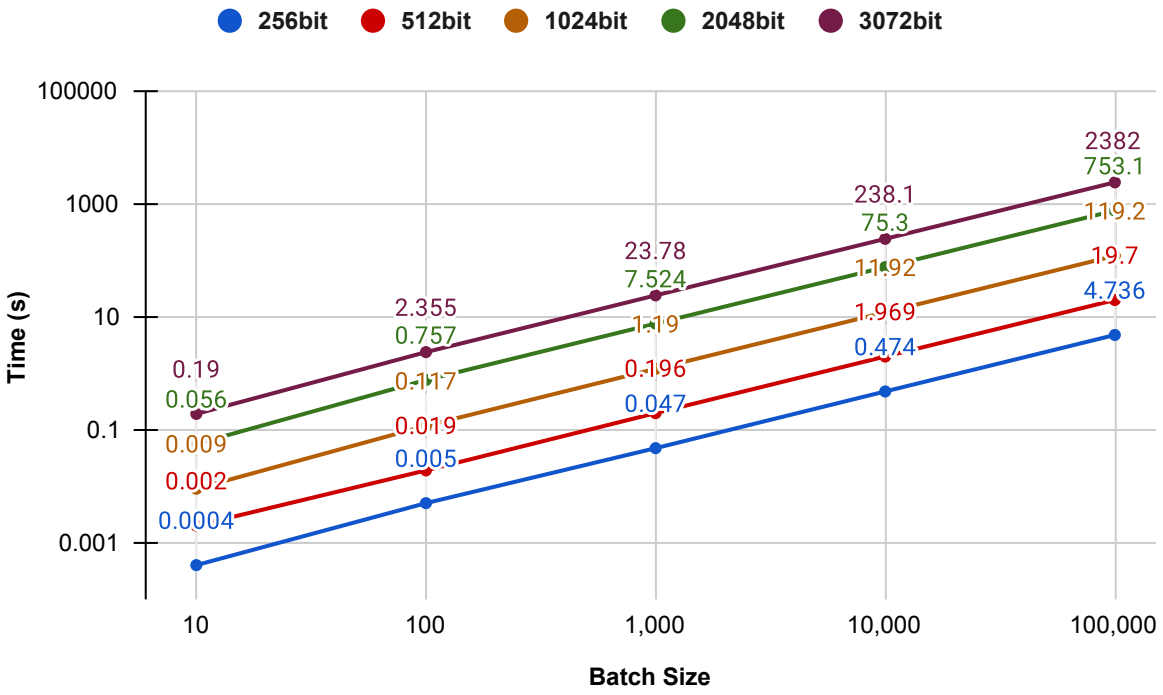

Figure 7.1: Ballot shuffling performance evaluation for different sizes of p (group modulus) and, therefore, different security levels. The figure shows the duration in seconds plotted against various batch sizes, *i.e.*, the number of votes in an operation. Both axis apply a logarithmic scale. Thus, a 45-degree slope represents linear growth.

Shuffle (Crypto Library). In Figure 7.1, the performance of the ballot shuffling operation, as implemented in the `crypto` package, is visualized for the different sizes of the group modulus p. It is clearly visible that an increase in group modulus p leads to a much more significant increase in runtime. For example, decreasing the size of p from 2048bit to 256bit, would allow for a performance increase of $\approx$ 150 times depending on the batch size. On the other hand, increasing p from 2048bit to 3072bit results in an increased runtime of $\approx$ 3.15 times. An option to allow decreasing the group modulus size while keeping or increasing the level of security is by performing all operations in a group $\mathbb{G}_q^*$ defined over an elliptic curve (EC). The performance gains are most likely less than proposed above as EC operations are computationally more expensive. Nevertheless, the performance improvements are still going to be significant enough to justify the effort. This idea has been addressed as a starting point for future work in Section 8.2.

ZKP Verification. The biggest bottleneck of this REV system are computationally expensive operations, such as ZKP verifications, that need to be performed by the PBB. As the authorities alternately produce new blocks in fixed time intervals and do not pool their resources, the DL can be perceived as a single unit of computing. The performance of the PBB in the current state depends on the underlying hardware of the authority responsible for authoring the current block. Therefore, the performance of the shuffle ZKP verification and partial decryption ZKP verification is further analysed.

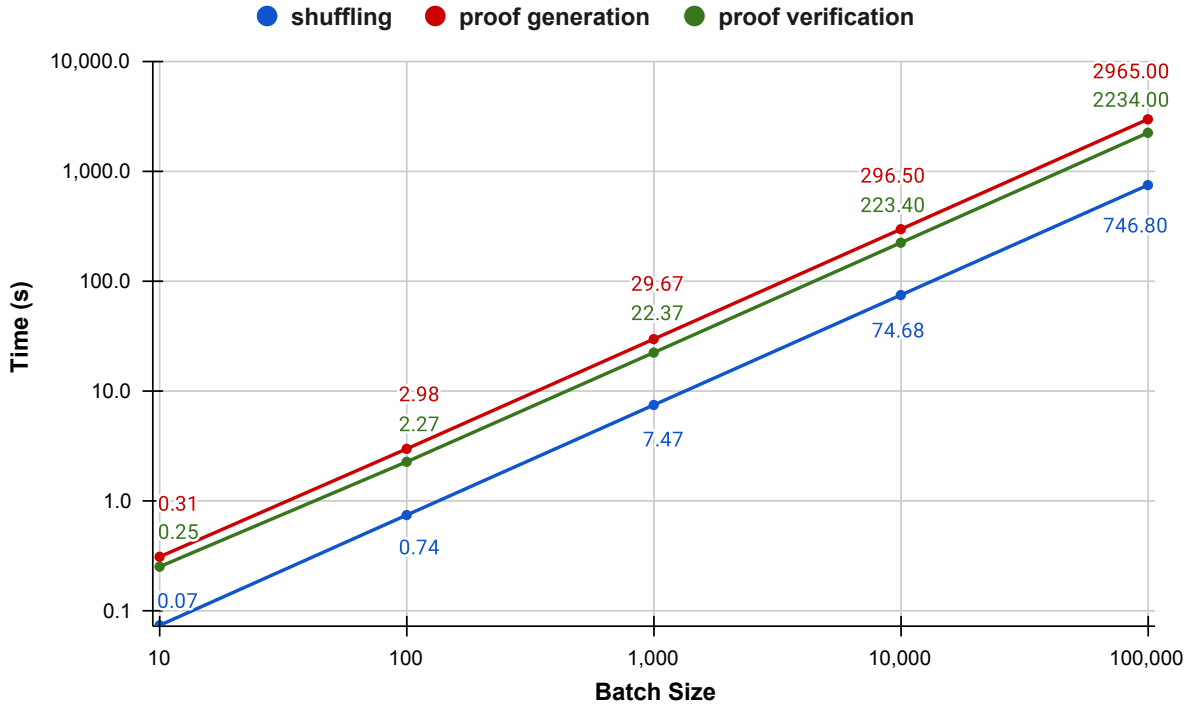

Figure 7.2: Mixnet performance evaluation for p of size 2048 *bits*. The figure shows the duration in seconds plotted against various batch sizes, *i.e.*, the number of votes in an operation. Both axis apply a logarithmic scale. Thus, a 45-degree slope represents linear growth.

Shuffle Proof. In Figure 7.2, the performance of the ballot shuffling, shuffle ZKP generation and verification are visualized for a single Substrate node. The runtime of the shuffling operation is comparable with the result of the `crypto` package benchmark for $p = 2048$ (see Figure 7.1). Additionally, shuffling as well as the shuffle proof generation are inherently parallelisable as these operations are performed offline. The work can be distributed across all sealers and an increase in ballots can be offset by an increase in sealers. Therefore, the performance of the shuffle proof verification is the only potential bottleneck as it is performed on the PBB by the DL network participant authoring the current block. In the current setup, this operation takes ≈ 22.3 *ms* per ballot which equates to 6.2 hours of shuffle ZKP verification for $1,000,000$ ballots. As of Substrate 3.0, runtime worker threads[1] have been added which allow a Substrate node to process mutually exclusive transactions, that do not alter the storage, in parallel. This is currently

[1] `https://github.com/paritytech/substrate/releases/tag/v3.0.0`

not implemented in the prototype but has been addressed as a starting point for future work in Section 8.2.

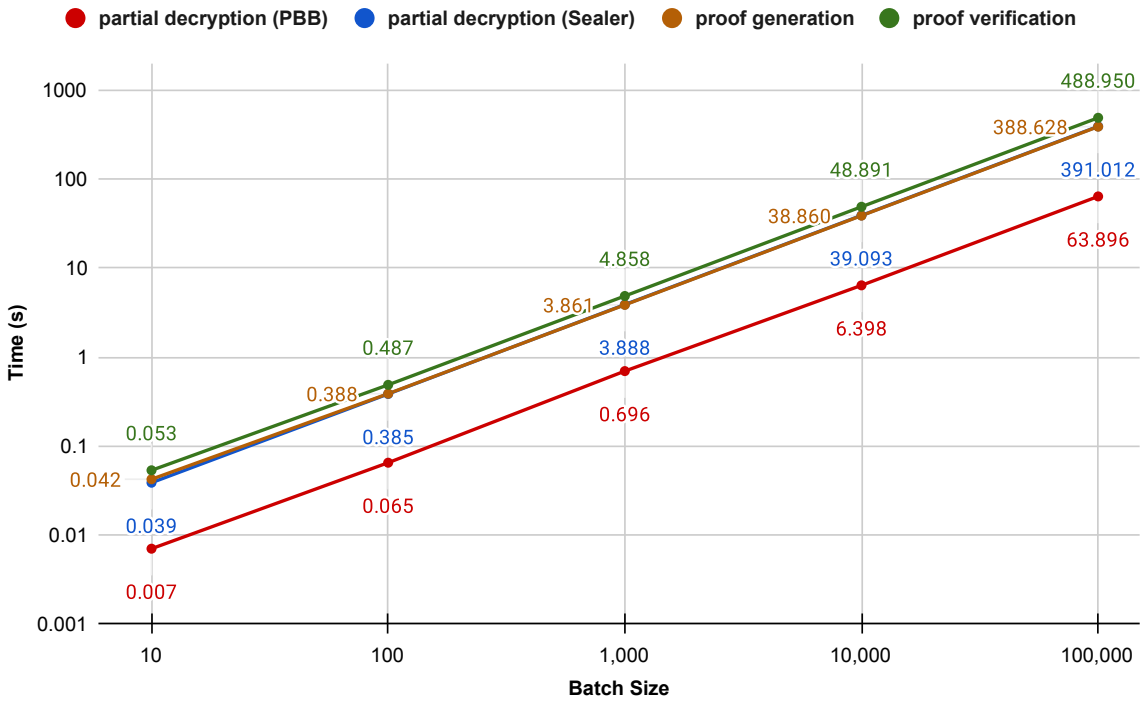

Figure 7.3: Partial decryption performance evaluation for p of size 2048 *bits*. The figure shows the duration in seconds plotted against various batch sizes, *i.e.*, the number of votes in an operation. Both axis apply a logarithmic scale. Thus, a 45-degree slope represents linear growth.

Partial Decryption Proof. In Figure 7.3, the performance of the the partial decryption (Sealer), the ZKP proof generation and verification as well as the partial decryption (PBB) are visualized for a single Substrate node. Similar to the ballot shuffling, the partial decryption (Sealer) and proof generation are performed offline and are therefore parallelisable. Thus, the proof verification and partial decryption (PBB) are the key operations, potentially posing a bottleneck, performed on the PBB by the DL network participant authoring the current block. In the current setup, the proof verification takes ≈ 4.8 *ms* per ballot which equates to 1.3 hours for 1 million ballots. Even though this does not directly pose a performance bottleneck, runtime worker threads could again be leverage to parallelize the proof verification. The performance of the partial decryption, performed on the PBB, does not pose a bottleneck as it only takes ≈ 0.6 *ms* per ballot which equates to 0.18 hours for 1 million ballots.

Chapter 8

Conclusion and Future Work

The last chapter of this work summarizes its achievements, draws the necessary and relevant conclusions and provides an outlook on what future research could focus on.

8.1 Conclusion

This work's main goal was to tackle the limited applicability of the existing Provotum projects to elections and to reduce the current ballot complexity to achieve nationwide scalability. For this, the possibilities of a mixnet-based remote electronic voting (REV) system were investigated. Additionally, due to its inherent integrity guarantee a distributed ledger (DL) was to be leveraged as public bulletin board (PBB) to ensure reliability, accountability and robustness on top of the commonly expected privacy and verifiability properties. The justification for evaluating mixnets in a REV setting is as they are advantageous over systems leveraging homomorphic encryption (HE) because they allow for multi-way elections and write-in candidates. Also, the computationally expensive zero-knowledge proof between the voter and the PBB ensuring ballot integrity is no longer necessary. On top of that, the remaining computationally heavy operations, such as ballot shuffling, are shifted from the voter to the voting infrastructure.

This work contributes a documented and fully functioning prototype implementation of the presented voting protocol. It extends prior work [66, 72] by enabling multi-way elections instead of only supporting yes/no votes, while retaining all privacy and verifiability properties. The prototype has been thoroughly evaluated and proven to scale sufficiently such that votes and elections up to one million ballots can be handled securely and efficiently. Finally, the development focused on ensuring extendablity, reusablity and ease of use of the prototype. Thus, pre-built Docker images and scripts to setup and demonstrate the capabilities with a single command are provided as well as a command line interface simplifying the interaction with the prototype and allowing to impersonate different stakeholders is available.

8.2 Future Work

The following section provides starting points for future work. It outlines different areas of the REV system as well as the voting protocol in which improvements are necessary or desired. The areas range from usability over technicalities to identity management.

8.2.1 Cast-as-Intended (CaI) Verifiability

The current protocol design does not support CaI verifiability for the layperson. A voter is required to have the cryptographic skills to audit the created ballot. Right now, the voter must understand how the ElGamal crypto system works and how an encryption is computed. This is infeasible in practice and requires a more user-friendly alternative. Thus, future work should investigate how CaI verifiability can be made suitable for the masses. Existing approaches make use of devices similar to the ones common in Internet banking applications. It would be interesting to explore smartphone based approaches as these devices are nowadays commonly available and already used as a replacement for the dedicated Internet banking devices.

8.2.2 k/n Distributed Key Generation

The current distributed key generation (DKG) algorithm requires a threshold of $t \leq n$, where $t = n$, authorities to be present both during the key generation as well as the cooperative decryption (CD). DKG and CD vastly improve the robustness and fairness of a REV system by distributing the trust across multiple nodes in the network and by making early decryption impossible. Nevertheless, the $t = n$ threshold bears its risks. It is no longer possible to tally a vote should a sealer loose access to its private key share or turn rogue and not contribute its partial decryptions. To mitigate this risk, alternative options such as Pedersen's DKG [90] not requiring all participants to be present at all times should be explored. Pedersen's DKG is based on every participant running simultaneously a Feldman's verifiable secret sharing (FVSS) [41] scheme as a dealer and, therefore, avoiding a single point of failure by only having a single dealer. FVSS is a widely known extension of Shamir's secret sharing [96] that allows to divide a secret value s across a set of participants t by splitting it into multiple parts. The scheme is based on the idea that the polynomial P is computed over $\mathbb{Z}_q^*$ and the secret s is defined as $s = P(0)$. Next, a set of points $P(0) = s, P(1), P(2), ..., P(t-1)$ is distributed privately among the participants privately. The polynomial P is chosen to be of degree $d = t - 1$ such that t participants, $t < n$ (*e.g.*, $n = 10, t = 7, d = 6$), can recover it using polynomial interpolation. Thus, recover the secret $s = P(0)$.

8.2.3 Technical Improvements

Elliptic Curve Cryptography. In order to improve the performance of the REV system while keeping the same level of security, an approach using elliptic curve cryptography

could be investigated. For example, a 256-bit elliptic curve (EC) public key cryptosystem provides roughly the same security as a 3248-bit ElGamal public key. The security of the ElGamal crypto system as defined in 3.4 is based on the difficulty of solving the discrete logarithm problem in the multiplicative, cyclic group $\mathbb{G}_q^*$. The group is defined over the finite field $\mathbb{Z}_p$, of positive integers modulo p, where $p = 2q+1$ and p and q are odd primes. Alternatively, the group $\mathbb{G}_q^*$ could instead be defined over an EC which itself is defined over the finite field $\mathbb{Z}_p$. In this alternative setting, the security is based on a similar problem of finding the discrete logarithm of a random EC point, instead of an integer. Due to the fact that efficiently computing a point multiplication is possible but finding the multiplicand provided a start- and end-point is mathematically intractable, the system is secure.

Substrate Runtime Worker Threads. The performance of a Substrate node is currently limited by the performance of a single core of its underlying hardware as all transactions are verified sequentially. This poses a problem for computationally expensive operations on the PBB such as shuffle- and partial decryption ZKP verification. To improve this performance, the recently introduced runtime worker threads[1] in Substrate could be leverage. This addition allows a Substrate node to parallelize the verification and processing of all mutually exclusive transactions, that do not alter the storage. To realize a performance improvement while adhering to the restrictions, the shuffle- and partial decryption ZKP transactions need to be restructured such that the proof verification can be performed in parallel while keeping the storage modification, *i.e.*, storing the shuffled ballots, sequential.

Mathematical Operations. Additionally, the performance of the underlying numeric library used to implement the mathematical operations has an impact on the performance of the actual algorithms [54]. The literature [53] proposes different techniques to speed up the process of shuffling ballots, generating and verifying shuffle ZKPs. Most of them are related to improving the speed of the computationally expensive `modExp` operations by improving the underlying algorithms for product and fixed-base exponentiation. Before such ideas are explored in future work, the numeric library used in this work, `num-bigint`[2], should be benchmarked and compared with other libraries known for high performance such as the *GNU Multiple Precision Arithmetic Library*[3] and its extension for modular exponentiation[4]. If deemed necessary and no other higher performing library already exists, ideas as proposed in literature could be explored to further improve the performance.

8.2.4 Identity Management

Apart from the inherently contradicting goals of privacy and verifiability, a third crucial property, the eligibility verification, exists that has not been investigated at all due to the limited time frame and scope of this work. Both previous Provotum projects [66, 72] have made attempts at improving the identity management situation. Nevertheless, the

[1] `https://github.com/paritytech/substrate/releases/tag/v3.0.0`
[2] `https://crates.io/crates/num-bigint`
[3] `https://gmplib.org`
[4] `https://github.com/verificatum/verificatum-gmpmee`

identity provider always remained a trusted third-party and responsibilities were simply shifted between different authorities. The problem itself is challenging and also somewhat contradicting in nature as its task is to uniquely identify potential voters without negatively affecting ballot secrecy while simultaneously ensuring that it is impossible for the identity provider to manipulate the result itself. One way or another, there is definitely further research and work required in this area.

Bibliography

[1] I. Abraham, B. Pinkas, and A. Yanai, "Blinder: Mpc based scalable and robust anonymous committed broadcast," Cryptology ePrint Archive, Report 2020/248, 2020, https://eprint.iacr.org/2020/248.

[2] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–35, 2018.

[3] B. Adida, "Helios: Web-based open-audit voting." in *USENIX security symposium*, vol. 17, 2008, pp. 335–348.

[4] S. T. Ali and J. Murray, "An overview of end-to-end verifiable voting systems," *CoRR*, vol. abs/1605.08554, 2016.

[5] S. D. Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain," in *ITASEC*, ser. CEUR Workshop Proceedings, vol. 2058. CEUR-WS.org, 2018.

[6] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand, "A guide to fully homomorphic encryption," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1192, 2015.

[7] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[8] U. N. G. Assembly, "Universal declaration of human rights," *UN General Assembly*, vol. 302, no. 2, 1948.

[9] A. Baraani-Dastjerdi, J. Pieprzyk, and R. Safani-Naini, *A practical electronic voting protocol using threshold schemes*. University of Wollongong. Department of Computing Science, 1994.

[10] S. Bayer and J. Groth, "Efficient zero-knowledge argument for correctness of a shuffle," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 263–280.

[11] J. Benaloh and D. Tuinstral, "Receipt-free secret-ballot elections (Extended abstract)," *Proceedings of the Annual ACM Symposium on Theory of Computing*, vol. Part F1295, pp. 544–553, 1994.

[12] D. Bernhard, O. Pereira, and B. Warinschi, "How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2012, pp. 626–643.

[13] D. Bernhard and B. Warinschi, "Cryptographic voting — a gentle introduction," in *Foundations of Security Analysis and Design VII*. Springer, 2013, pp. 167–211.

[14] D. Boneh, "The decision diffie-hellman problem," in *International Algorithmic Number Theory Symposium*. Springer, 1998, pp. 48–63.

[15] A. Caforio, L. Gasser, and P. Jovanovic, "A decentralized and distributed e-voting scheme based on verifiable cryptographic shuffles," 2017. [Online]. Available: https://www.epfl.ch/labs/dedis/wp-content/uploads/2020/01/report-2017-2-andreacaforio-evoting.pdf

[16] Canton of Geneva. E-Voting System: CHVote 1.0. [Online]. Available: https://republique-et-canton-de-geneve.github.io/chvote-1-0/index-en.html

[17] ——. E-Voting System: CHVote 2.0. [Online]. Available: https://chvote2.gitlab.io/state

[18] B. Carter, K. Leidal, D. Neal, and Z. Neely, "Survey of Fully Verifiable Voting Cryptoschemes," no. May, pp. 1–12, 2016.

[19] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–88, 1981.

[20] ——, "Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 330 LNCS, pp. 177–182, 1988.

[21] D. Chaum, D. Das, F. Javani, A. Kate, A. Krasnova, J. De Ruiter, and A. T. Sherman, "cmix: Mixing with minimal real-time asymmetric cryptographic operations," in *International Conference on Applied Cryptography and Network Security*. Springer, 2017, pp. 557–578.

[22] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *CRYPTO*, ser. Lecture Notes in Computer Science. Springer, 1992, vol. 740, pp. 89–105.

[23] B. Chevallier-Mames, P.-A. Fouque, D. Pointcheval, J. Stern, and J. Traoré, "On Some Incompatible Properties of Voting Schemes," in *Lecture Notes in Computer Science*, 2010, vol. 6000 LNCS, pp. 191–199.

[24] R. Cramer, "Modular design of secure yet practical cryptographic protocols," Ph.D. dissertation, 1996.

[25] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," *European Transactions on Telecommunications*, vol. 8, no. 5, pp. 481–490, 1997.

[26] E. Cuvelier, O. Pereira, and T. Peters, "Election verifiability or ballot privacy: Do we need to choose?" Cryptology ePrint Archive, Report 2013/216, 2013, https://eprint.iacr.org/2013/216.

[27] D. Das, S. Meiser, E. Mohammadi, and A. Kate, "Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 108–126.

[28] D. Demirel and J. van de Graaf, "A publicly-veriable mix-net with everlasting privacy towards observers," Cryptology ePrint Archive, Report 2012/420, 2012, https://eprint.iacr.org/2012/420.

[29] Der Schweizerische Bundesrat, "Verordnung über die politischen Rechte (VPR) (vom 24. Mai 1978 (Stand 1. Juli 2019))," 1978. [Online]. Available: https://www.admin.ch/opc/de/classified-compilation/19780105/index.html

[30] Die Bundesversammlung der schweizerische Eidgenossenschaft, "Die Bundesverfassung der Schweizerischen Eidgenossenschaft (vom 18. April 1999 (Stand am 23. September 2018))," 2018. [Online]. Available: https://www.admin.ch/opc/de/classified-compilation/19995395/201809230000/101.pdf

[31] Die Bundesversammlung der schweizerischen Eidgenossenschaft, "Bundesgesetz über die politischen Rechte (vom 17. Dezember 1976 (Stand am 1. November 2015))," 1976. [Online]. Available: https://www.admin.ch/opc/de/classified-compilation/19760323/index.html

[32] Die schweizerische Bundeskanzlei (BK), "Verordnung der BK über die elektronische Stimmabgabe (VEleS) (vom 13. Dezember 2013 (Stand am 1. Juli 2018))," 2013. [Online]. Available: https://www.admin.ch/opc/de/classified-compilation/20132343/index.html

[33] ——, "Erläuternder Bericht zur Anpassung der Verordnung der BK über die elektronische Stimmabgabe (VEIeS) (vom 30. Mai 2018)," 2018. [Online]. Available: https://www.bk.admin.ch/bk/de/home/politische-rechte/e-voting/versuchsbedingungen.html

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

[35] E. Dubuis, "E-Demokratie: E-Voting," in *Handbuch E-Government*. Wiesbaden: Springer Fachmedien Wiesbaden, 2019, pp. 1–14. [Online]. Available: http://link.springer.com/10.1007/978-3-658-21596-5%5F39-1

[36] M. El Laz, B. Grégoire, and T. Rezk, "Security analysis of elgamal implementations," in *17th International Conference on Security and Cryptography*. SCITEPRESS-Science and Technology Publications, 2020, pp. 310–321.

[37] T. Elgamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[38] E. F. Hao, "Schnorr Non-interactive Zero-Knowledge Proof," RFC Editor, Tech. Rep., 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8235

[39] P. Fauzi and H. Lipmaa, "Efficient culpably sound nizk shuffle argument without random oracles," in *Cryptographers' Track at the RSA Conference*. Springer, 2016, pp. 200–216.

[40] P. Fauzi, H. Lipmaa, J. Siim, and M. Zając, "An efficient pairing-based shuffle argument," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 97–127.

[41] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, 1987, pp. 427–438.

[42] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 263. Springer, 1986, pp. 186–194.

[43] C. Fontaine and F. Galand, "A survey of homomorphic encryption for nonspecialists," *EURASIP J. Information Security*, vol. 2007, 2007.

[44] A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections," in *International Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 1992, pp. 244–251.

[45] J. Furukawa and K. Sako, "An efficient scheme for proving a shuffle," in *Annual International Cryptology Conference*. Springer, 2001, pp. 368–387.

[46] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *J. Cryptol.*, vol. 20, no. 1, pp. 51–83, 2007.

[47] J. P. Gibson, R. Krimmer, V. Teague, and J. Pomares, "A review of e-voting: the past, present and future," *Annales des Télécommunications*, vol. 71, no. 7-8, pp. 279–286, 2016.

[48] K. Gjøsteen, "The Norwegian Internet Voting Protocol," in *Lecture Notes in Computer Science*, 2012, vol. 7187 LNCS, pp. 1–18. [Online]. Available: http://link.springer.com/10.1007/978-3-642-32747-6%5F1

[49] K. Gjøsteen, T. Haines, and M. R. Solberg, "Efficient mixing of arbitrary ballots with everlasting privacy: How to verifiably mix the ppatc scheme," Cryptology ePrint Archive, Report 2020/1331, 2020, https://eprint.iacr.org/2020/1331.

[50] J. Groth, "Efficient maximal privacy in boardroom voting and anonymous broadcast," in *International Conference on Financial Cryptography*. Springer, 2004, pp. 90–104.

[51] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu, "Aggregatable distributed key generation," Cryptology ePrint Archive, Report 2021/005, 2021, https://eprint.iacr.org/2021/005.

[52] R. Haenni, R. E. Koenig, P. Locher, and E. Dubuis, "Chvote system specification 3.2," *IACR Cryptology ePrint Archive*, vol. 2017, p. 325, 2017.

[53] R. Haenni and P. Locher, "Performance of shuffling: Taking it to the limits," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 369–385.

[54] R. Haenni, P. Locher, and N. Gailly, "Improving the performance of cryptographic voting protocols," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 272–288.

[55] R. Haenni, P. Locher, R. Koenig, and E. Dubuis, "Pseudo-code algorithms for verifiable re-encryption mix-nets," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 370–384.

[56] T. Haines, S. J. Lewis, O. Pereira, and V. Teague, "How not to prove your election outcome," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 644–660.

[57] T. Haines and J. Müller, "Sok: Techniques for verifiable mix nets," in *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*. IEEE, 2020, pp. 49–64.

[58] T. Haines, R. Gore, and B. Sharma, "Did you mix me? formally verifying verifiable mix nets in electronic voting," Cryptology ePrint Archive, Report 2020/1114, 2020, https://eprint.iacr.org/2020/1114.

[59] F. S. Hardwick, A. Gioulis, R. N. Akram, and K. Markantonakis, "E-voting with blockchain: An e-voting protocol with decentralisation and voter privacy," in *iThings/GreenCom/CPSCom/SmartData*. IEEE, 2018, pp. 1561–1567.

[60] S. Hauser and R. Haenni, "Implementing broadcast channels with memory for electronic voting systems," *JeDEM-eJournal of eDemocracy and Open Government*, vol. 8, no. 3, pp. 61–79, 2016.

[61] C. Hébant, D. H. Phan, and D. Pointcheval, "Linearly-homomorphic signatures and scalable mix-nets," in *IACR International Conference on Public-Key Cryptography*. Springer, 2020, pp. 597–627.

[62] S. Heiberg, A. Parsovs, and J. Willemson, "Log analysis of estonian internet voting 2013–2014," in *Lecture Notes in Computer Science*, 2015, vol. 9269, pp. 19–34. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-22270-7%5F2https://eprint.iacr.org/2015/1211.pdfhttp://link.springer.com/10.1007/978-3-319-22270-7%5F2

[63] R. Henry, A. Herzberg, and A. Kate, "Blockchain access privacy: Challenges and directions," *IEEE Security Privacy*, vol. 16, no. 4, pp. 38–45, 2018.

[64] L. Hirschi, L. Schmid, and D. Basin, "Fixing the achilles heel of e-voting: The bulletin board," Cryptology ePrint Archive, Report 2020/109, 2020, https://eprint.iacr.org/2020/109.

[65] M. Hirt, "Receipt-free K-out-of-L voting based on elgamal encryption," in *Towards Trustworthy Elections, New Directions in Electronic Voting*, ser. Lecture Notes in Computer Science, D. Chaum, M. Jakobsson, R. L. Rivest, P. Y. A. Ryan, J. Benaloh, M. Kutylowski, and B. Adida, Eds., vol. 6000. Springer, 2010, pp. 64–82. [Online]. Available: https://doi.org/10.1007/978-3-642-12980-3_3

[66] A. Hofmann, "Security Analysis and Improvements of a Blockchain-based Remote Electronic Voting System," Ph.D. dissertation, University of Zurich, 2020.

[67] H. Jonker, S. Mauw, and J. Pang, "Privacy and verifiability in voting systems: Methods, developments and trends," *Computer Science Review*, vol. 10, pp. 1–30, 2013.

[68] H. Jonker and J. Pang, "Bulletin boards in voting systems: Modelling and measuring privacy," in *ARES*. IEEE Computer Society, 2011, pp. 294–300.

[69] C. D. M. Jr., "Report of the national workshop on internet voting: issues and research agenda," in *Proceedings of the 2000 National Conference on Digital Government Research, {DG.O} 2000, Los Angeles, CA, USA, May 15-17, 2000*, 2000. [Online]. Available: http://dl.acm.org/citation.cfm?id=1123096

[70] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic elections," in *Towards Trustworthy Elections*, ser. Lecture Notes in Computer Science. Springer, 2010, vol. 6000, pp. 37–63.

[71] A. Kate and I. Goldberg, "Distributed key generation for the internet," in *2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE, 2009, pp. 119–128.

[72] C. Killer, B. Rodrigues, E. J. Scheid, M. Franco, M. Eck, N. Zaugg, A. Scheitlin, and B. Stiller, "Provotum: A blockchain-based and end-to-end verifiable remote electronic voting system," in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, 2020, pp. 172–183.

[73] C. Killer, B. Rodrigues, R. Matile, E. John Scheid, and B. Stiller, "Design and implementation of cast-as-intended verifiability for a blockchain-based voting system," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20. Association for Computing Machinery, 03 2020, pp. 286–293.

[74] S. Kremer, M. Ryan, and B. Smyth, "Election verifiability in electronic voting protocols," in *ESORICS*, ser. Lecture Notes in Computer Science, vol. 6345. Springer, 2010, pp. 389–404.

[75] R. Krimmer, *A Structure for New Voting Technologies: What They Are, How They Are Used and Why: Bridging the Gap Between Information Systems Research and Practice*. Springer, 01 2019, pp. 421–426.

[76] R. Küsters, T. Truderung, and A. Vogt, "Accountability: definition and relationship to verifiability," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 526–535.

[77] ——, "Verifiability, privacy, and coercion-resistance: New insights from a case study," in *2011 IEEE Symposium on Security and Privacy*. IEEE, 2011, pp. 538–553.

[78] R. Kusters, T. Truderung, and A. Vogt, "Clash attacks on the verifiability of e-voting systems," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 395–409.

[79] S. J. Lewis, O. Pereira, and V. Teague, "How not to prove your election outcome The use of non-adaptive zero knowledge proofs in the Scytl-SwissPost Internet voting system, and its implications for decryption proof soundness," pp. 1–11, 2019.

[80] Y. Liu and Q. Wang, "An e-voting protocol based on blockchain," *IACR Cryptology ePrint Archive*, vol. 2017, p. 1043, 2017.

[81] R. Matile, "Cast-as-Intended Verifiability in Blockchain-based Electronic Voting for Swiss National Elections," Ph.D. dissertation, University of Zurich, 2018. [Online]. Available: https://www.merlin.uzh.ch/contributionDocument/download/11375

[82] R. Matile and C. Killer, "Privacy, Verifiability, and Auditability in Blockchain-based E-Voting," Ph.D. dissertation, University of Zurich, 2018. [Online]. Available: https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/mp-raphael-christian.pdf

[83] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in *Financial Cryptography*, ser. Lecture Notes in Computer Science, vol. 10322. Springer, 2017, pp. 357–375.

[84] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[85] C. A. Neff, "A verifiable secret shuffle and its application to e-voting," in *Proceedings of the 8th ACM conference on Computer and Communications Security*, 2001, pp. 116–125.

[86] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Business & Information Systems Engineering*, vol. 59, no. 3, pp. 183–187, 2017.

[87] R. Oppliger, *Contemporary cryptography*, ser. Artech House computer security series. Artech House, 2005.

[88] C. Park, K. Itoh, and K. Kurosawa, "Efficient anonymous channel and all/nothing election scheme," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1993, pp. 248–259.

[89] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '91. Berlin, Heidelberg: Springer-Verlag, 1991, p. 129–140.

[90] ——, "A threshold cryptosystem without a trusted party," in *EUROCRYPT*, ser. Lecture Notes in Computer Science. Springer, 1991, vol. 547, pp. 522–526.

[91] O. Pereira and V. Teague, "Report on the SwissPost-Scytl e-voting system , trusted-server version," pp. 1–41, 2019.

[92] J. Proos and C. Zalka, "Shor's discrete logarithm quantum algorithm for elliptic curves," *arXiv preprint quant-ph/0301141*, 2003.

[93] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[94] K. Sako and J. Kilian, "Receipt-free mix-type voting scheme - A practical solution to the implementation of a voting booth-," *Lecture Notes in Computer Science*, vol. 921, pp. 393–403, 1995.

[95] C. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[96] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, Nov. 1979.

[97] F. Shirazi, S. Neumann, I. Ciolacu, and M. Volkamer, "Robust electronic voting: Introducing robustness in civitas," in *2011 International Workshop on Requirements Engineering for Electronic Voting Systems*. IEEE, 2011, pp. 47–55.

[98] W. D. Smith, "Cryptography meets voting," *Compute*, vol. 10, p. 64, 2005. [Online]. Available: http://www.hit.bme.hu/~buttyan/courses/BMEVIHI5316/Smith.Crypto%5Fmeets%5Fvoting.pdf

[99] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman, "Security Analysis of the Estonian Internet Voting System," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14*, no. May. New York, New York, USA: ACM Press, 2014, pp. 703–715. [Online]. Available: https://dl.acm.org/citation.cfm?id=2660315http://dl.acm.org/citation.cfm?doid=2660267.2660315

[100] O. Spycher and M. Volkamer, *E-Voting and Identity*, ser. Lecture Notes in Computer Science, A. Kiayias and H. Lipmaa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, vol. 7187, no. September. [Online]. Available: http://link.springer.com/10.1007/978-3-642-32747-6

[101] W. Stallings, *Network Security Essentials: Applications and Standards, 6th Edition*. Pearson Education, 2017.

[102] M. H. Suwito, Y. Ueshige, and K. Sakurai, "Evolution of bulletin board & its application to e-voting – a survey," Cryptology ePrint Archive, Report 2021/047, 2021, https://eprint.iacr.org/2021/047.

[103] J. Svensson and R. Leenes, "E-voting in Europe: Divergent democratic practice," *Information Polity*, vol. 8, no. 1-2, pp. 3–15, 2003. [Online]. Available: http://content.iospress.com/articles/information-polity/ip000023

[104] B. Terelius and D. Wikström, "Proofs of restricted shuffles," in *International Conference on Cryptology in Africa*. Springer, 2010, pp. 100–113.

[105] W. Wang, D. T. Hoang, P. Hu, Z. Xiong, D. Niyato, P. Wang, Y. Wen, and D. I. Kim, "A survey on consensus mechanisms and mining strategy management in blockchain networks," *IEEE Access*, vol. 7, pp. 22 328–22 370, 2019.

[106] D. Wikström, "Verificatum mix-net (vmn)," Sep. 2020. [Online]. Available: https://www.verificatum.org/html/product_vmn.html

[107] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger." *Ethereum Project Yellow Paper*, 2014.

[108] K. Wüst and A. Gervais, "Do you Need a Blockchain?" in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, no. i. IEEE, 6 2018, pp. 45–54.

Abbreviations

BS	Ballot Secrecy
CaI	Cast-as-intended
CaR	Counted-as-recorded
CR	Coercion-Resistance
DDH	Decisional Diffie-Hellmann
DKG	Distributed Key Generation
DL	Distributed Ledger
E2E-V	End-To-End Verifiability
El-V	Eligibility Verifiability
IV	Individual Verifiability
NIZKP	Non-Interactive Zero Knowledge Proof
PoA	Proof of Authority
PoW	Proof of Work
PBB	Public Bulletin Board
RaC	Recorded-as-cast
REV	Remote Electronic Voting
RF	Receipt-Freeness
TTP	Trusted Third Party
UP	Unconditional Privacy
UV	Universal Verifiability
ZKP	Zero Knowledge Proof

List of Figures

3.1 Pseudo-Code Permutation Algorithm 14

3.2 Pseudo-Code Shuffle Algorithm 15

3.3 A generic Σ protocol 18

3.4 Key Generation Proof 19

3.5 Decryption Proof 20

3.6 Designated Verifier Re-Encryption Proof 21

3.7 Shuffle Proof Generation 23

3.8 Shuffle Proof Verification 24

3.9 Algorithm to retrieve n independent generators for vote id_{vote}. 25

3.10 Algorithm to generate permutation commitment for permutation ψ. 25

3.11 Algorithm to generate n challenge values for a set of public input values $\mathbf{e}, \tilde{\mathbf{e}}, \mathbf{c}, pk$. 26

3.12 Algorithm to generate commitment chain for the permuted public challenges $\tilde{\mathbf{u}}$. 26

5.1 Provotum Stakeholders 33

5.2 Provotum's Voting Protocol Phases 35

5.3 **Pre-Voting Phase** - Registration Step - Create Vote & Store Questions . . 36

5.4 **Pre-Voting Phase** - Key Generation Step - Key Pair Generation, Proof Generation & Verification, Storage of Public Key Share 36

5.5 **Pre-Voting Phase** - Key Generation Step - Combine Public Key Shares to Create Vote Public Key 37

5.6 **Voting Phase** - Ballot Generation, Ballot Randomization & Casting Steps . 38

5.7 **Post-Voting Phase** - Ballot Shuffling Step - Ballot Shuffling, Proof Generation/Verification and Storage 39

5.8 **Post-Voting Phase** - Tallying Step - Ballot Shuffling, Partial Decryption . . 40

5.9 **Post-Voting Phase** - Results Step - Combine Partial Decryptions, Decrypt and Reveal Plaintexts 40

6.1 Provotum Prototype Packages 41

6.2 `node/pallets/mixnet/src/lib.rs`: the main entry point of the mixnet pallet. 42

6.3 The default message shown by the `provotum-cli`. For each component, a different subcommand provides the built-in operations. 43

6.4 The figure shows the function header and documentation for `encrypt_encode` which encodes and encrypts a message/vote such that it is possible to perform additive homomorphic operations on the ciphertext. The code is found in the following file: `crypto/src/encryption.rs` 44

6.5 The figure shows the function documentation and implementation for `partial_decrypt_b` which decrypts the component b of an ElGamal encryption **e** = (a,b) given a partially decrypted **e**. The code is found in the following file: `crypto/src/encryption.rs` 45

7.1 Ballot shuffling performance evaluation for different sizes of p (group modulus) and, therefore, different security levels. The figure shows the duration in seconds plotted against various batch sizes, *i.e.*, the number of votes in an operation. Both axis apply a logarithmic scale. Thus, a 45-degree slope represents linear growth. 55

7.2 Mixnet performance evaluation for p of size 2048 *bits*. The figure shows the duration in seconds plotted against various batch sizes, *i.e.*, the number of votes in an operation. Both axis apply a logarithmic scale. Thus, a 45-degree slope represents linear growth. 56

7.3 Partial decryption performance evaluation for p of size 2048 *bits*. The figure shows the duration in seconds plotted against various batch sizes, *i.e.*, the number of votes in an operation. Both axis apply a logarithmic scale. Thus, a 45-degree slope represents linear growth. 57

List of Tables

7.1 Visualization of the privacy properties across all three Provotum projects. . 47

7.2 Visualization of the verifiability properties across all three Provotum projects. 49

7.3 Visualization of the practical properties across all three Provotum projects. 51

7.4 Comparison of cryptographic operations across the existing Provotum projects in terms of `modExp`. 52

7.5 The performance difference of the two message encoding strategies (see Section 3.4.3) for p of size 2048 $bits$ and $q = \frac{p-1}{2}$. 54

Appendix A

Installation Guidelines

The prototype's source code, all packages developed as part of this thesis and the pre-built Docker container images can be found in the Provotum GitHub organization in the `provotum-mixnet` repository at `https://github.com/provotum/provotum-mixnet`.

A `README` is provided for the main repository as well as for each package. The intent of this document is to simplify familiarizing oneself with the project and to provide answers to frequently reoccurring questions. Instructions for running a demo of the prototype are also contained. A Docker Compose[1] script is provided alongside to simplify the technical aspects of running a demo. To ensure compatibility, Docker version 20.10 and Docker Compose 1.27.4 or higher are required.

[1] `https://docs.docker.com/compose`

Appendix B

Contents of the CD

The following items are contained in an archive accompanying this thesis:

- A PDF file of this report.
- The LaTeX source code of this report.
- The images of this report.
- The source code of the prototype.
- The source code of the benchmarks.
- The raw and processed results from the different benchmarks.