University of Zurich^UZH

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

Design and Implementation of Unconditional Everlasting Privacy in Blockchain-based Remote Electronic Voting

Claude Simon Müller
Zürich, Switzerland
Student ID: 11-067-675

Supervisor: Christian Killer, Bruno Rodrigues
Date of Submission: April 30, 2020

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Züurich, 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 digitalization of democratic processes has attracted much attention in recent years. For example, spreading and signing a petition via the internet is now easier than ever. Likewise, electronic voting is of interest to many countries and organizations. Ongoing research in this area for over two decades shows that it is no simple task to transfer the analogous act of casting a ballot and vote-tallying to the digital realm. The opacity of digital systems requires ways of verifying that every valid vote is included in the vote counting and no manipulations happened. This property is called verifiability. At the same time, the system must ensure the voter's privacy. Although many countries have made the step to electronic voting (*e.g.,* in the form of direct recording devices placed at polling stations), only in rare cases ballots can be cast remotely in so-called Remote Electronic Voting (REV) systems. Research has proposed several REV protocols offering varying levels of ballot privacy and verifiability, but practical implementations are few. Therefore, this thesis focuses on the design and implementation of a REV system based on a protocol with the specific property of unconditional privacy. This privacy notion does not depend on trust in a central authority nor the intractability of an underlying mathematical problem. To satisfy the protocol's minimal trust assumptions, our implementation applies blockchain as an enabler of transparent, tamper-proof, and decentralized platforms. In particular, we make use of Tendermint and Cosmos-SDK to construct an append-only bulletin board and design a voting system prototype around it that implements the protocol. The prototype highlights pitfalls and necessary considerations that only become clear through the transfer from theory to practice. It shows that, in combination with blockchain, the voting protocol's central issue becomes scalability, an issue to be overcome in future research.

Kurzfassung

Die Digitalisierung demokratischer Prozesse hat in den letzten Jahren viel Aufmerksamkeit auf sich gezogen. Zum Beispiel ist das Verbreiten und Unterzeichnen von Petitionen über das Internet einfacher als je zuvor. Gleichermassen sind elektronische Abstimmungen für viele Länder und Organisationen von Interesse. Die seit über zwei Jahrzehnten laufenden Forschungen auf diesem Gebiet zeigen, dass es kein Leichtes ist, die analoge Stimmabgabe und Stimmenauszählung in die digitale Welt zu überführen. Die Undurchsichtigkeit digitaler Systeme erfordert, dass man überprüfen kann, ob jede gültige Stimme in die Stimmenzählung einbezogen wurde und keine Manipulationen stattgefunden haben. Diese Eigenschaft wird als Verifizierbarkeit bezeichnet. Gleichzeitig muss das System die Privatsphäre des Wählers gewährleisten. Obwohl viele Länder den Schritt zu elektronischen Abstimmungen gewagt haben (z.B. in Form von digitalen Abstimmungsgeräten in Wahllokalen), können Stimmzettel nur in seltenen Fällen in sogenannten Remote Electronic Voting (REV) Systemen aus der Ferne abgegeben werden. Die Forschung hat verschiedenste REV-Protokolle hervorgebracht, die in unterschiedlichem Ausmass Vertraulichkeit und Verifizierbarkeit bieten. Nur wenige wurden auch in einer Implementation umgesetzt. Diese Arbeit konzentriert sich auf den Entwurf und die Implementierung eines REV-Systems basierend auf einem Protokoll mit der speziellen Eigenschaft der bedingungslosen Vertraulichkeit. Diese Auffassung von Vertraulichkeit hängt weder vom Vertrauen in eine zentrale Behörde noch von der Unlösbarkeit eines mathematischen Problems ab. Um die minimalen Vertrauensannahmen des Protokolls zu erfüllen, bedient sich die Implementierung an der Blockchain-Technologie, welche Transparenz, Dezentralisierung und Schutz vor Manipulation verspricht. Insbesondere verwenden wir Tendermint und Cosmos-SDK zur Konstruktion eines elektronischen Anschlagbretts, um welches ein Prototyp eines Abstimmungssystems entworfen wird, das das genannte Protokoll implementiert. Der Prototyp zeigt Herausforderungen und notwendige Überlegungen, die erst durch die Übertragung von der Theorie in die Praxis deutlich werden. Es stellt sich heraus, dass in Kombination mit Blockchain, die Skalierbarkeit zum zentrale Problem des Abstimmungsprotokolls wird. Ein Problem, das es in zukünftigen Forschungsarbeiten zu lösen gilt.

Acknowledgments

Several people have accompanied me throughout this thesis and kept me on track and sane. Their support made this work possible.

Special thanks go to Christian Killer who, as my supervisor, always helped me with valuable feedback, suggestions, discussions, and inputs to bring me forward step by step. I want to thank Prof. Dr. Burkhard Stiller for the opportunity to work on this thesis; Philipp Locher, for the discussions on his voting protocol; Lucas, who was writing his thesis in parallel and was always available for a rant about current issues; and finally, Anouchka, for her encouragements.

Contents

Chapter 1

Introduction

Voting and electoral processes form the foundations of democracies. They allow participants to vote and elect representatives. The means for vote casting vary greatly and have evolved over time [Kri19]. With today's digitalization and the widespread adoption of the Internet, a new remote voting channel is available. Casting a vote over the Internet promises more convenience, potentially a more verifiable and transparent voting process, and a reduction of efforts in setting up and conducting polls with physical labor [KDKV+18]. However, with it comes a whole new set of problems that need to be solved for new voting systems to continue fulfilling familiar properties like voter privacy [JMP13].

Active research on the topic of Remote Electronic Voting (REV) was conducted for already more than two decades, giving rise to a multitude of voting protocols [JMP13]. Most work in this area heavily relies on cryptographic tools for ensuring the desirable properties of REV systems. Over the years, these properties were formalized under the most prominent concepts of privacy and verifiability. Privacy is intuitively about keeping the voter's choice a secret while verifiability makes a voting system transparent. The two sound like a contradiction. How can one verify the results of an election while not knowing who voted what? However, solutions for this dilemma exist and are part of existing, operational voting systems. Though, different solutions make different assumptions and support the properties to different degrees [JMP13].

Verifiability can be provided at the level of each voter and on a global level. In the first case, every voter can rest assured that their vote was correctly recorded, and the second case gives anyone the possibility to check if all valid votes have been correctly included in the final voting result [JMP13]. Privacy can be divided into basic, everlasting, and unconditional privacy. The basic instance relies on the assumption of computational intractability and a trusted central authority. Privacy is broken if the assumption ceases to hold or the authority cheats. Everlasting privacy removes the intractability assumption, and unconditional privacy goes even further by also removing the need for a trusted authority [Loc16]. The former is essential for ensuring that votes stay secret not only at the time of the poll but also in the future. The latter is of interest in scenarios where the authority cannot be trusted, or no central authority is available, *e.g.,* a consortium of industry participants. Instead of consulting a third party to take on the role of the authority, the consortium could run the voting system in a trustless, decentralized manner.

Most protocols proposed in the research literature do not make the step to unconditional privacy. Instead, they rely on a trusted entity or base their privacy on a secret shared between multiple trusted entities [Loc16]. Furthermore, many protocols stay a theoretical construct and are not transferred into practice. The underlying assumptions therefore, never undergo a reality check.

With many countries showing interest in REV and participating in pilot projects, we think it is necessary to have more research on the practicality of existing REV protocols [E-V]. Therefore, this thesis explores the practical feasibility of decentralized voting systems with the properties of verifiability and, in particular, unconditional privacy as achieved by the protocol proposed by Locher and Haenni in [LH15] and [Loc16]. As a modern building block of distributed systems, we target blockchain as a significant component of decentralized voting systems. Blockchains allow us to forgo the need for a central authority and instead spread responsibility and control over multiple entities. Furthermore, blockchain replicates data and secures it against modifications, making it ideal for storing ballots that should not be manipulated by anyone [KRMS+20].

The thesis' goals are (1) the design of a suitable system for blockchain-based REV, where (2) the implementation of the unconditional privacy protocol presented in [LH15] is achieved, and (3) the evaluation of said protocol, examining it with a focus on scalability and blockchain applicability.

1.1 Thesis Outline

The rest of this thesis is structure as follows. To establish common ground from which to assess REV protocols, Chapter 2 conveys an understanding of the voting system properties that appear in the research literature. The Chapter also presents the cryptographic primitives underlying the REV protocol that is at the core of this work and gives a short introduction to blockchains. In Chapter 3, several REV systems and protocols are presented, starting with examples for state-level voting and then progressing through academic work with a focus on ballot privacy. Chapter 4 introduces the REV protocol that this thesis is based on and then documents the design process of a system implementing the protocol. It devises the desirable properties for a suitable blockchain platform and examines the properties of several established blockchains, leading to a specific choice. It further discusses the protocol's practical ramifications and proposes a system architecture. Chapter 5 documents the actual implementation, showing which parts of the implementation cover which parts of the protocol. The evaluation in Chapter 6 reflects on the translation of the protocol's properties into the prototype and analyzes the system's performance and scalability. Chapter 7 summarizes the achievements and findings of this thesis and suggests many possible future research and implementation efforts.

Chapter 2

Background

Electronic voting is a broad term covering multiple types of voting systems that all have the use of electronic devices in common. Even some aspects of postal voting systems are part of this field. The Office for Democratic Institutions and Human Rights of the OSCE defined the term New Voting Technologies (NVT), which includes any technology that makes use of ICT for casting and counting votes. These technologies are further divided, into more specific applications, like ballot-scanning technologies, direct-recording electronic voting systems, or internet voting [OSC13].

This work only addresses internet voting, or more formal, Remote Electronic Voting (REV). A particularity of REV is that voters cast ballots in an uncontrolled environment. While polling stations can specify a fixed setup for every voter, REV has to deal with the problem of the untrusted voter platform and an untrusted and open network like the internet.

This chapter introduces the system properties that the research literature uses to specify how well a voting protocol can deal with issues as the ones mentioned above. Furthermore, the necessary cryptographic primitives for the rest of this work are introduced, and the essential concepts of blockchains are stated.

2.1 Electronic Voting Systems Desiderata

Over the years, the academic literature on Electronic Voting (EV) systems has introduced a set of desirable properties for such systems. The definition of these properties can differ depending on the author. Therefore, they are discussed and defined here. This work does not extend the theoretical aspects of voting systems and, therefore, does not provide any formal definitions. The goal is to set up definitions as a basis for the discussion of different REV systems. [JMP13] traces the emergence of different properties throughout the literature and forms the basis for the definitions shown here.

2.1.1 Verifiability

In traditional voting systems, like paper-based postal voting, the voter cannot verify that her vote was correctly included in the vote counting. She would have to trace the ballot through the process of placing it into the ballot box (or post box), emptying the box, anonymizing the ballots and tallying the votes, which is unrealistic for a large number of voters. The same problem exists in electronic voting, where even observable, physical processes are absent. Mechanisms are needed that allow a voter to verify if the voting system recorded her ballot correctly, and allow anyone to verify if the vote-tallying includes all valid votes [CPP13]. These two aspects of the verifiability property are called individual verifiability and universal verifiability, respectively. They are sometimes divided into more specific aspects. The following defines all commonly used verifiability properties.

Definition 1 (Individual Verifiability). *A voting system has the individual verifiability property if a voter can verify that her ballot is in the recorded set of ballots and contains her intended vote [JMP13].*

Definition 2 (Universal Verifiability). *A voting system has the universal verifiability property if anyone can verify that the voting result was tallied correctly, meaning that all valid ballots were included with their intended vote [SK95].*

Definition 3 (End-to-End Verifiability). *End-to-end verifiability is defined similarly to individual verifiability and universal verifiability, but divided into three more specific properties. See [BRRS+15] for an in-depth definition.*

1. *Cast-as-Intended verifiability requires that a voter can verify that her ballot contains the intended vote. For example, this is not obvious if votes are encrypted before sending them.*
2. *Recorded-as-Cast verifiability demands that a voter can verify that the voting system received and stored her ballot correctly. For example, in the case of a public list of votes, the voter can consult the list and check if it contains her vote.*
3. *Counted-as-Recorded verifiability says that anyone can verify the correctness of the voting result, meaning that it includes all the recorded and valid ballots.*

Definition 4 (Eligibility Verifiability). *A voting system has the eligibility verifiability property if anyone can verify that the voting result only contains votes from eligible voters, and only includes one vote per voter [KRS10].*

Both end-to-end verifiability and the combination of individual and universal verifiability cover similar aspects. Research usually uses one of the two, but not both simultaneously.

Individual and cast-as-intended verifiability fixes the problem of the untrusted voter platform in REV. They are necessary if the voter's platform cannot be controlled and trusted, which is the case if the voter should be able to cast a ballot from personal electronic devices. The two notions assure that the voter is at least able to detect that her vote was compromised, most probable by malicious code on the voting device [HKLD18].

2.1.2 Privacy

Privacy in voting is essential for keeping a voter anonymous and her decision secret, protecting her from coercion, and prohibit vote selling. REV puts the voter and the voter client into an uncontrolled environment, *e.g.,* her home and her electronic devices. Thus, REV protocols have to deal with the possibility that the voter client is compromised by malware, and the voter is observed or coerced while casting her vote. Privacy in this domain is usually split into Ballot Privacy (BP), coercion-resistance, and receipt-freeness [JMP13]. Furthermore, BP can be defined with different assumptions in mind and is further divided into basic BP, everlasting BP, perfect BP, and unconditional privacy, all described in the following.

Definition 5 (Ballot Privacy). *A voting system with BP does not allow any coalition of computationally bounded voters or outside observers to determine for whom a voter voted.*

The assumption of computational boundedness implies that BP is based on some computational intractability assumption. If in the future, this assumption becomes false and the system stores ballots linked to voter IDs, BP is lost. It is, therefore, also called computational privacy [MN06]. The definition does assume trust in voting authorities. That is, a voting system with BP is not safe against conspiring voting authorities.

Definition 6 (Everlasting Ballot Privacy). *A voting system with everlasting BP does not allow even a computationally unbounded coalition of voters or outside observers to determine for whom a voter voted [MN06].*

Everlasting BP enhances the basic BP in that the privacy does not depend on computational intractability but has to be information-theoretic [MN06]. Thus, the secrecy of the ballot is preserved even if a computationally unbounded adversary appears in the future. Again, the definition does assume trust in voting authorities. That is, a voting system with everlasting BP is not safe against conspiring voting authorities.

Definition 7 (Perfect Ballot Privacy). *A voting system with perfect BP does not allow any computationally bounded entity to determine for whom a voter voted [KY02].*

Under this definition, even if a set of voting authorities conspire, they are not able to assign votes to voters. Therefore, a voting scheme with this property does not require any trust in the voting authorities nor any other party. However, this privacy property relies on computational intractability and therefore is not everlasting. If everlasting and perfect BP are combined, we speak of unconditional PB.

Definition 8 (Unconditional Ballot Privacy). *A voting system with unconditional BP does not allow anyone, not the voters, nor outside observers, nor the voting authorities, be they computationally unbounded or not, to determine for whom a voter voted [Loc16].*

Note that none of the above BP hide information that can be derived from the final voting result or a partial result calculated by a subgroup of voters. For example, if all but one voter conspire and calculate the result of only their votes, they will immediately know the vote of the excluded voter by comparing their result with the total result. Such a conspiracy is unlikely but impossible to prevent in a system that openly publishes the election results.

The BP definitions are not concerned with the information about a voter taking part in an election or not. Hiding this information is defined under Participation Privacy.

Definition 9 (Participation Privacy). *A voting scheme with participation privacy does not allow anyone with access to only the public voting data to determine if a voter has participated in an election or not [KTV15].*

It is evident that a voting system with participation privacy also offers ballot privacy since it is a contradiction to know a voter's vote but not know if she took part.

Definition 10 (Receipt-Freeness). *A receipt-free voting system does not allow the voter to obtain a receipt showing her voting choices. That is, the voter does not receive nor can she produce anything that proves her choice to someone else [JN06].*

Receipt-freeness is the leading property that prevents vote-buying. Without a receipt and with intact vote privacy, a buyer has to trust the voter that she indeed voted as agreed. Different strengths of receipt-freeness can be defined, where the strongest one denies the construction of a receipt even if the voter reveals her private key material. Receipt-freeness is arguably more critical in REV than in paper-based voting because the availability of digital receipts improves the scalability of vote-buying activities [Loc16].

Definition 11 (Coercion-Resistance). *In a coercion-resistant voting system, a coercer cannot determine nor influence a voter's vote (implying receipt-freeness) and is additionally not able to force vote randomization or abstention from voting [JCJ10].*

The example of a private voting booth provides coercion-resistance in that a coercer cannot decide if the voter has cast the imposed vote. Receipt-freeness can be part of this property because coercion-resistance is lost if the voter is provided with a receipt. However, giving the voter multiple opportunities for voting, the need for receipt-freeness can be removed. In such a scheme, after previously being coerced, a voter could recast her vote without the coercer's presence. In the strongest notion, even the divulgence of private key material does not allow coercion [JCJ10].

2.1.3 Others

Definition 12 (Fairness). *A voting scheme that offers fairness prohibits anyone from calculating a partial tally before the end of the voting period [Gro04].*

The reasoning behind the fairness property is that knowledge about intermediary election results can be used to distort the final results. The above definition prevents everyone (voters and the voting authorities) from calculating partial results. Depending on the voting system, this is relaxed and the definition does not include the voting authorities

[Gro04]. Note that fairness is a property required only during the voting period. In contrast to privacy, it cannot be broken after an election has taken place.

Electronic voting systems can also be described in the context of standard non-functional software system requirements. Such properties are not unique to voting systems, but specific definitions are given here nonetheless.

Definition 13 (Scalability). *A voting system is scalable if it can support large electorates without incurring unsustainable demands to data storage and computational power.*

For example, if the generation of a ballot becomes too demanding for user devices when increasing the electorate size, the system is not scalable. Depending on the use case a voting system is intended for, scalability plays an important role. If it aims at state-wide popular votes, scalability is of great importance. If it is used in board room elections, scalability is neglectable. The scalability becomes evident by analyzing the runtime and space requirements of the algorithms used in a voting scheme. In practice, bad scalability could be mitigated by splitting the electorate into manageable subsets.

Definition 14 (Robustness). *A voting system is robust if failing system components or malicious attackers cannot easily disrupt its availability and consistency.*

Robustness is mostly a practical property, only to a lesser extend reflected in theoretical work on voting systems. If, in theory, a protocol is based on distributed trust among several authorities, then this hints that a practical implementation thereof might achieve robustness towards malicious voting authorities. It is up to the practical realization to make sure that components assumed by protocols, like communication channels, are implemented robustly.

2.1.4 Discussion

The difficulty of reaching the above properties depends very much on the trust assumptions of a voting system. If the system assumes trust in the voting authority or a third party (*e.g.,* a ballot printing service), it requires less effort to fulfill the properties. However, that assumption creates a central point of failure and possibly malicious activity. Delegating trust to various voting authorities mitigates this problem in that malicious activity now requires several conspiring authorities. In Definition 8 the privacy property is stated without dependence on such trust, which is the most potent notion of privacy. However, in the end, the use case determines the required trust assumptions and strength of the voting system's property. I.e., not every voting system requires all the above properties.

Some of the properties seem to be in contradiction. For example, receipt-freeness seems to be easily obtainable by merely omitting any receipts or other feedback to the voter about his vote. However, this approach collides with the individual verifiability property, which intuitively requires some information about a vote being returned to the voter. Another example is voter eligibility, demanding that everyone can verify the eligibility of participating voters while at the same time, participation privacy requires voter identities to stay anonymous.

The above properties have been defined for electronic voting systems and are not all applicable to non-electronic systems. For example, in the Swiss postal voting system, universal verifiability is not possible. Voters cannot verify the final voting result. Voters have to trust that the postal service and the voting authorities do their work honestly.

2.1.5 Public Bulletin Board

This section does not deal with a voting system property per se, but rather a concept that many REV protocols rely on. That concept is called Public Bulletin Board (PBB), and it serves as storage for election parameters, ballots, and other data that arise in an election. The PBB is often described as a public, append-only broadcast channel with memory [HH16]. Public availability is usually needed for the universal verifiability property. The append-only property of the PBB implies immutability. Any data written to the board must never be removed or modified, or such changes must at least be detectable by any observer.

Definition 15 (Public Bulletin Board). *The PBB is a reliable broadcast channel with memory that everyone can read from, and election participants can write to [Loc16]. It has the following properties:*

- *New data is appended in sequence, i.e., the arrival time of messages is represented by their ordering on the PBB.*
- *The contents of the PBB cannot be altered or removed.*
- *The PBB must be robust against failures.*
- *New data is verified before appending it.*

The use of the word participants in the definition hints that only authorized parties can write to the PBB. However, more precisely, this means that the PBB has to make sure no one can publish data in the name of someone else [Loc16]. E.g., an attacker cannot cast a ballot in the name of another participating voter. The last property mentioned in the definition reinforces this. It forbids that any unchecked data is posted to the PBB.

2.2 Cryptography Preliminaries

Electronic voting shows a wide use of cryptographic tools for ensuring confidentiality, integrity, and other properties of voting protocols. This section introduces the cryptography-related subjects that are fundamental to the voting protocol implemented in this work and also often found in other protocols.

2.2.1 Group Theory for Cryptography

In mathematics, the study of group theory lays essential foundations for many achievements in cryptography. That is, significant parts of today's asymmetric cryptography are based on particular groups studied in group theory.

Definition 16 (Group). *A group $(G, \cdot)$ is a mathematical construct consisting of a set of elements G and a binary operation $(\cdot)$ that takes two elements $a, b \in G$ and produces a third element $c = a \cdot b \in G$ while satisfying the four axioms of (1) closure, (2) associativity, (3) identity, and (4) invertibility [Sma16].*

1. *Closure requires that any element formed by applying the group operation to two elements of G must again be an element of G.*

2. *The group operation must be associative. I.e, for all $a, b, c \in G$ it must hold that $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.*

3. *There must be a unique element $e \in G$ (the identity element) for which $a \cdot e = a$ holds. Meaning that applying the identity element to any other element of G results in the same element.*

4. *For every element $a \in G$ there exists an element $b \in G$ that is the inverse of a, denoted as a^{-1}. The inverse applied to the element a results in the identity element, namely $a \cdot b = e$.*

Groups provide an abstraction of what we intuitively use in everyday life when adding together two numbers. The group we are using in that example are the integers under addition $(\mathbb{Z}, +)$. It is straightforward to show that it fulfills all the required properties of a group. Groups have an order which quantifies the number of their elements [Lan05]. A group's order can be finite or infinite. In the example of $(\mathbb{Z}, +)$, the order is infinite.

If a group contains one or more elements that can be used to generate all other elements of the group, the group is called cyclic.

Definition 17 (Cyclic Group). *A cyclic group is a group $(G, \cdot)$ which is generated by an element $a \in G$. That is, every element of G can be obtained by repeatedly applying the group operation to a or its inverse. The element a is called a generator of the group [Sma16].*

The integers over addition are again an example for a cyclic group. The element 1 is its generator. Every other element can be generated by adding 1 or -1 multiple times.

Another example of cyclic groups is the so-called multiplicative group of integers modulo n, where n is a prime number. These groups use multiplication modulo n as their group operation. They contain integers from the set $\{1, \ldots, n-1\}$ and the group operation is applied as $a \cdot b \pmod{n}$.

Definition 18 (Multiplicative Group of Integers modulo n). *A multiplicative group of integers modulo n is a cyclic group where the set G consists of non-negative integers from $\{1, \dots, n-1\}$ that are coprime to n and the group operation is multiplication modulo n [Sma16]. Conversely put, the integers coprime to n form a group under multiplication modulo n. In this work such a group is denoted as $\mathbb{Z}_n^*$ [LH15]. Other popular notations are $(\mathbb{Z}/n\mathbb{Z})^\times$ or $\mathbb{Z}/n\mathbb{Z}$.*

The definition points out that only the integers coprime to n are elements of the group, not all integers in the range $[1, n-1]$. Only then are the group axioms fulfilled. If n is chosen to be a prime number, then all integers in $[1, n-1]$ are coprime with n and, therefore, the group order is n.

To be useful in cryptography, groups need to show specific properties, one of them being the decisional Diffie-Hellman assumption.

Definition 19 (Decisional Diffie-Hellman Assumption). *Given a multiplicative cyclic group $\mathbb{G}$ with order q and generator g, and two uniformly and independently chosen elements $a, b \in \mathbb{Z}$, the decisional Diffie-Hellman assumption (DDH) states that for g^a and g^b, the value g^{ab} cannot be distinguished from a random element in $\mathbb{G}$ [Sma16].*

In other words, if given g^a, g^b and g^{ab} one cannot decide if g^{ab} was actually formed from a and b or some other random element $c \in \mathbb{G}$ This assumption implies that it is infeasible to calculate the logarithm of g^a or g^b. Otherwise, it would be possible to produce $a = log_g(g^a)$ and then compare $(g^b)^a$ to g^c. This underlying, weaker assumption is called the discrete logarithm assumption. The discrete logarithm over groups of integers is an integer $x = log_g(a)$ such that $g^x = a$, where a and g are also integers of the group.

Definition 20 (Discrete Logarithm Assumption). *The discrete logarithm assumption (DL) holds for a group $\mathbb{G}$ if there is no algorithm that can find $x = log_g(a)$ in polynomial time where g is a generator and a any element of $\mathbb{G}$ [Sma16].*

In contrast to the logarithm over $\mathbb{R}$, no efficient algorithm is known that can solve the discrete logarithm problem in general [Sma16]. That is, there are some groups for which the discrete logarithm assumption is believed to hold. Schnorr groups are one of them. They are subgroups of $\mathbb{Z}_p^*$.

Definition 21 (Subgroup). *Given a group $\mathbb{G}$ with element set G. The set H forms a subgroup $\mathbb{H}$ of $\mathbb{G}$ if it is a subset of G and fulfills the group axioms under the group operation of $\mathbb{G}$ [Lan05].*

Definition 22 (Schnorr Group[1]). *Schnorr groups are subgroups $\mathbb{G}_q$ of $\mathbb{Z}_p^*$ with prime order q and modulus p. They are generated by choosing p, q and k such that $p = qk + 1$, with p and q being prime [KL07]. To find a generator of the subgroup choose any h in the range $1 < h < p$ until you find one such that $h^r \not\equiv 1 \pmod{p}$. From h one can calculate the generator $g = h^k \pmod{p}$ of the subgroup [Nat13].*

[1]We do not know of any reference that actually uses the name Schnorr group for this kind of group. But, the same kind of group is called Schnorr group on Wikipedia (see `https://en.wikipedia.org/wiki/Schnorr_group`) and we use this name for convenience.

Note that even though the modulus p of the subgroup $\mathbb{G}_q$ is prime, the order of the group is not p but q, *i.e.,* not all integers from $[0, p-1]$ are elements of $\mathbb{G}_q$. If x is an element of $\mathbb{G}_q$ is determined by $0 < x < p$ and $x^q \equiv 1 \pmod{p}$.

Throughout this work, we refer to two groups $\mathbb{G}_p$ and $\mathbb{G}_q$, that are at the basis of the implemented voting protocol. For both, the DDH assumption is required. Therefore, we construct them as Schnorr groups. For $\mathbb{G}_p$, we denote the modulus with o and the order with p. For $\mathbb{G}_q$, the modulus is denoted as p and the order q. The two groups are linked by p being the order of the former and the modulus of the latter.

Even though it is believed that the DL and DDH hold in Schnorr groups, this is not true for any choice of the groups modulus p and order q [KL07]. If p and q are chosen too small, brute-forcing the discrete logarithm problem could break the security assumptions. Therefore, it is essential to choose these parameters according to a specified security level.

Definition 23 (Security Level). *The security level or strength is a number denoting the amount of work required to break a cryptographic system. It is usually specified in bits [Len04].*

For some algorithms, like the AES, the security level coincides with the length of the secrete key that is input to the algorithm. In others, the input key size needs to be much bigger to achieve the same security strength. In RSA, for example, a key size of 3072 is associated with a security level of 128 because methods exist that make breaking the system more efficient than using a brute-force attack on the key [Bar16]. At the time of writing, according to the National Institute of Standards and Technology (NIST), a security level of 80 is not considered to be save anymore. Instead, at least 112 should be used [Bar16].

2.2.2 Commitment Schemes

Commitment schemes are used to commit to a value without actually revealing it at the time of commitment but possibly revealing it later. They are useful when a party must be prohibited to change a value they have committed to, but the value should also stay private at the time of commitment [Sma16]. Instead of giving a general description of commitment schemes, we introduce a specific instance, namely the Pedersen commitment scheme [Ped92].

Pedersen commitments require the usage of a prime-order subgroup $\mathbb{G}_q$ of $\mathbb{Z}_p^*$ that fulfills the DL assumption (*e.g.,* a Schnorr group). The commitment to a value $v \in \mathbb{Z}_q$ is defined as

$$com_p(r, v) = g^r h^v$$

where $r \in_R \mathbb{Z}_q$, *i.e.,* it is randomly chosen from $\mathbb{Z}_q$, and g and h are generators of $\mathbb{G}_q$. The value $c \leftarrow com_p(r, v)$ is published while r and v are kept private. The commitment can be opened by simply publishing the two values v and r. The receiver of the values then calculates $c' \leftarrow com_p(v, r)$ and accepts if $c' = c$ [Ped92].

The scheme can be extended and used to commit to multiple values at the same time. A commitment to values $u_1, \ldots, u_n \in \mathbb{Z}_q$ is defined as

$$com_p(r, u_1, \ldots, u_n) = g^r h_1^{u_1} \ldots h_n^{u_n}$$

with $r \in_R \mathbb{Z}_q$ and $g, h_1, \ldots, h_n$ being generators of $\mathbb{G}_q$ [Loc16].

The Pedersen commitment scheme is binding and perfectly hiding. It is binding because based on the DL assumption, it is hard for the originator to find values r' and v' such that $com_p(r', v') = com_p(r, v)$. It is perfectly hiding because $c \leftarrow com_p(r, v)$ does not reveal any information about the committed value v. The hiding property does not depend on any computational hardness assumption but is true because many values v' and r' can open the commitment c. An attacker can, therefore, not decide which v' and r' are the right ones [Loc16].

2.2.3 Zero-Knowledge Proofs

In electronic voting systems, trust is an issue. Even if the voting authorities are trusted, the voter and their client should not be trusted. However, at the same time, privacy requirements demand that information about the voter and her ballot are not revealed. For example, if the voter sends an encrypted ballot, she has to prove that the ballot is in a valid format without revealing the actual vote itself. This dilemma can be solved with a zero-knowledge proof (ZKP).

Definition 24 (Zero-knowledge Proof). *A ZKP is an interactive protocol in which a prover tries to prove to a verifier that a particular statement is correct but without revealing any knowledge about the statement apart from the fact that it is correct [Sma16].*

The formal definition of a ZKP is omitted here. It can be found, for example, in [GMR89]. A ZKP system needs to be sound, complete, and zero-knowledge [Sma16].

Definition 25 (Soundness). *A ZKP system is sound if, given a false statement, a prover cannot convince the verifier that the statement is correct with more than a small probability.*

Definition 26 (Completeness). *A ZKP system is complete if, given a true statement, the verifier will be convinced by the statement. That is, the prover will succeed conveying the correctness of her statement.*

Definition 27 (Zero-knowledge). *A ZKP system is zero-knowledge if, given a true statement, the verifier does not learn anything but the correctness of the statement.*

ZKPs can be used to prove that one has knowledge of something without revealing that knowledge. This is then called a zero-knowledge proof of knowledge [KL07]. Taking the Pedersen commitment scheme as an example, a prover can prove knowledge of the opening (v, r) of a commitment c. Merely revealing the values v and r is also a proof of knowledge but not a zero-knowledge one. A ZKP of knowledge provides the same proof but without

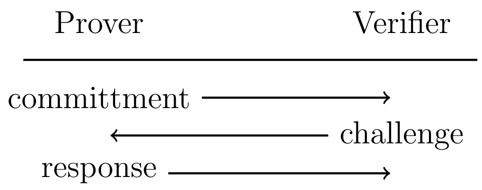


Figure 2.1: The three steps of a Σ-protocol

revealing the values. Such proofs are often formulated in a Σ-protocol. A Σ-protocol is a three-move protocol of the form shown in Figure 2.1. The protocol begins with the prover sending a message to the verifier. By sending the message, the prover commits to its contents. The verifier then creates and sends a random value that the prover cannot predict. It serves as a challenge to which the prover then has to create a response. The response is based on the challenge and the values sent in the commitment phase. If the prover has the knowledge, the verifier will accept the response [Dam02].

Because the parties taking part in an electronic voting system will not be available all the time to serve as provers in an interactive ZKP, it is desirable to make the proof non-interactive. With a Non-Interactive Zero-Knowledge Proof (NIZKP), the prover can generate a proof once and make it available to many verifiers even after going offline. For this to work, the verifier's random challenge needs to be replaced. The Fiat-Shamir heuristic offers a solution here [FS87]. In the formal definition, the new source of the challenge is a so-called random oracle. A random oracle can be queried with some input and responds with a random response. The response is always the same if queried repeatedly with the same input. In practice, cryptographic hash functions are used for such oracles. The input to the hash function is the data from previous steps of the ZKP protocol [Sma16]. The prover must not be able to predict the challenge resulting from his inputs. Otherwise, the proof loses the soundness property.

We will use the notation NIZKP[$\cdot$] to refer to a non-interactive ZKP or a proof transcript thereof. For example, for a proof proving knowledge of a discrete logarithm, we write NIZKP[$x : g^x$]. The protocol implemented in this work uses three specific NIZKPs shortly introduced in the following.

Set Membership Proof

With a set zero-knowledge membership proof, a prover can prove that a publicly known set U contains some value u without revealing the value. Multiple such proofs have been proposed in the past [Loc16]. The one used in this work was initially presented in [BG13] and is based on the evaluation of a polynomial. Input to the proof is a polynomial $P(X) = \sum_{i=0}^{n} a_i X^i \in \mathbb{Z}_p[X]$ and a commitment $com(r, u)$ to the value u. $\mathbb{Z}_p[X]$ is called a polynomial ring. The coefficients of a polynomial in this ring are elements of $\mathbb{Z}_p$, where $\mathbb{Z}_p$ is the ring of integers modulo prime p. The polynomial $P(X)$ is computed from the set $U = u_0, \ldots, u_n$ as $P(X) = \prod_{i=0}^{n}(X - u_i)$. Therefore, the polynomial evaluates to zero $P(u_i) = 0$ for every $u_i \in U$. In the proof transcript the prover has to prove that she knows the opening of the commitment $com(r, u)$ to value u and that $P(u) = 0$.

We refer to [BG13] and [Loc16] for the whole proof description.

Proof of Known Representation of a Committed Value

The term representation in the proofs name refers to the discrete logarithm representation $v_1, \dots, v_n$ of a value $u = h_1^{v_1} \cdots h_n^{v_2}$. The $v_i \in \mathbb{Z}_q$ are the representation of $u \in \mathbb{G}_q$. $\mathbb{G}_q$ is a cyclic group with order q and generators $h_1, \dots, h_n$ and $\mathbb{Z}_q$ is the set of integers modulo q. If u was public we could construct a proof similar to a proof of known discrete logarithm [Sch89]. But in our case u is not public but only available in committed form $com_p(r, u) = g_1^r g_2^u$, where g_1 and g_2 are generators of another cyclic group G_p with order p and $r \in_R \mathbb{Z}_p$. If we plugin $u = h_1^{v_1} \cdots h_n^{v_2}$ we get $g_1^r g_2^{h_1^{v_1} \cdots h_n^{v_2}}$ which is why this proof is also called proof of knowledge of double discrete logarithm.

We refer to [ASM10] and [Loc16] for the whole proof description.

Proof of Known Equality of Discrete Logarithms

The simplest proof is called a proof of known equality of discrete logarithms or preimage equality proof. It is similar to a proof of known discrete logarithm (see [Sch89]) but involves two values. For example, prove knowledge of discrete logarithm x of g^x and x' of $h^{x'}$ while also showing that $x = x'$. Or prove knowledge of the opening (r, x) of a commitment $h_1^r h_2^x$ and a discrete logarithm x' of $g^{x'}$ while showing that $x = x'$.

2.3 Blockchain

Blockchain is an integral part of this work. Therefore, we introduce a few blockchain concepts here. At its core, blockchain is a data structure that chains blocks of data together by cryptographic means, thereby making it impossible to change one block without affecting all subsequent blocks. The chaining is based on hashes that are stored in the blocks and generated with a cryptographic hash function. Each block stores the hash of the previous block as a pointer, thereby forming a chain. Modifying the data contained in a block changes the block's hash, which breaks the link to the next block. The blockchain data structure is shared in a distributed system with many nodes maintaining a copy of it [ZXDC+17].

Another vital part of a blockchain is its consensus mechanism. It is responsible for forming new blocks and establishing consensus on the state of the chain. Since the original idea behind blockchain is the abandonment of a trusted third party, the consensus mechanism makes sure that no single entity can govern the creation of new blocks. Instead, multiple participants have to find a consensus on which block to add next [WHHM+19]. The consensus mechanism should ensure the safety and liveness of the blockchain network. Safety means that each participant receives the same blocks in the correct order and, in the end, has the same view on the blockchain's state. Liveness means that the network does not come to a halt, meaning each node will eventually receive every transaction and block, and block production continues [CV17].

Over the last few years, different consensus mechanisms have evolved, beginning with Proof-of-Work (PoW) in Bitcoin[2]. PoW is based on a cryptographic puzzle that needs to

[2]`https://bitcoin.org`

be solved to be able to propose a new block. The node that solves the puzzle first can create the block and broadcast it to the network. By seeing the solution for the puzzle the other network participants have proof that the originator put in the work. The effort needed to create valid blocks is what stops attackers from rewriting the blockchain history [WHHM+19].

Other prominent consensus mechanisms are Proof-of-Stake (PoS) and Proof-of-Authority (PoA). Both renounce the intense computational work required by PoW. In PoS, any network participant can put some value at stake (usually a native cryptocurrency), giving him the power to propose new blocks. The idea is that the staked value is proof that the owner has something at stake and therefore has the interest to keep the network running correctly [ZXDC+17].

PoA reduces the proof to one of identity, meaning that only participants that have been given authorization can produce blocks. The consensus nodes still have to communicate with each other to reach a consensus on which block gets created next, which means that no single entity can produce blocks at will [Szi17].

Chapter 3

Related Work

The academic literature has proposed numerous variants of REV schemes. For a few of them, an implementation exists, and even fewer are in practical use. The application domain ranges from state-wide popular votes (*e.g.,* sVote [Mau19]) to small boardroom elections (*e.g.,* [KY02]). This chapter presents a selection of REV systems and protocols. The focus is on privacy properties, which are of special interest in the protocol implemented in this work.

3.1 REV Systems for Popular Votes

Estonia introduced a REV system in 2005 as one of the first states worldwide. It serves as a well-established example of large-scale, real-world REV systems and has been used in many nation-wide elections since its introduction. Over the years, it has been scrutinized and improved and fulfills more desirable properties today, than when it was first deployed [ABJO+10; SFDK+14; HPW15].

A REV system has to be integrated into an already established landscape of voting systems and can not just replace the old system. The same applies to the Estonian system, which needed and still needs to work side-by-side with polling stations. Voters are allowed to cast their votes in the polling stations or use the REV system. In case of duplicate voting, *i.e.,* casting via the internet and in the voting booth, the vote cast in the polling station overrides the remotely cast vote. Multiple voting is a countermeasure against vote-buying and is also available in the REV system itself, giving voters the ability to cast multiple ballots, each overriding the previous one. The idea is simple: a vote buyer or coercer cannot be sure if a voter will override her original vote with a new vote [Maa04].

Another difficulty of introducing any new voting and electoral process is voter identity management. Because Estonia already had established a digital citizen identity system, it eased the introduction of the REV system. Voters can use their usual digital identity (*e.g.,* identity card and smart card reader) to authenticate, verify their eligibility, and cast a vote [Vin12].

The verifiability and privacy properties of the Estonian voting system are not optimal from the perspective of the academic literature on REV. The system started without individual and universal verifiability. Those were only added beginning in 2013 and later [GKTP16]. From a privacy perspective, the system is based on trust in the voting authorities. The measures to preserve BP are described as follows: "At no point should any party of the system be in possession of both the digitally signed e-vote and the private key of the [central voting] system." [Com05]. The operators have to follow strict procedures to ensure privacy and consistency [SFDK+14]. The Estonian example shows that a REV system used for nation-wide votes does not need to have all the properties defined in Section 2.1 to be successful in practice.

The example of Switzerland shows a much more conservative approach to REV adoption than Estonia. Although the first trials were already conducted in 2003 in the canton of Geneva, at the time of writing, Switzerland does not have an accepted nation-wide internet voting system. One reason for this is the federalist structure of Switzerland. Every canton has authority over its voting systems. Therefore, multiple projects are being developed in parallel, and cantons have formed consortia, each developing a joint system [SGMP+15].

In 2013 the Federal Chancellery of Switzerland defined clear guidelines on the properties an internet voting system has to fulfill in order to be admitted for votes and elections of different sizes. The regulation was published as the ordinance on electronic voting (updated in 2018) [Bun13]. The Swiss Post REV system is one project that set out to reach those properties and make internet voting available to cantons. It is developed by Scytl[1], a Spanish company with international reputation and focus on REV systems. The project's source code was disclosed to a selected audience of security researchers in February 2019 [Mau19]. In-depth security analysis did show significant errors in the source code and differences between code and protocol specification [PT19]. In March, the Swiss Post announced that the trials are put on hold until serious vulnerabilities are fixed [Pos19].

The other prominent Swiss internet voting system aimed at communal, cantonal, and federal votes is CHVote, developed by the canton of Geneva. Geneva started its efforts in 2001 and conducted first trial votes on its system in 2003 [HKLD18]. Due to concerns about the security of the voter's platform and new requirements for verifiability and transparency, a second project, CHVote 2.0, was started in January 2017. It was designed to fulfill the full expansion stage described in the federal chancellery's ordinance on electronic voting, meaning that it would be allowed to serve 100% of the cantonal and federal electorate. However, development was halted in December 2018. At the same time, the discontinuation of the production-ready CHVote 1.0 was announced. Both due to cost reasons [HKLD17].

3.2 Implemented REV Protocols from Reasearch

A few smaller REV systems have emerged from research and find real-world applications on a limited scale. They are no less important than nationwide systems since their findings

[1] `https://www.scytl.com/`

provide the basis for improving those larger systems.

Helios[2] appears prominently in the literature and has been around since 2008 when the first paper on it was published [Adi08]. The system has been further developed with multiple publications documenting the improvements.

Helios provides privacy by asymmetrically encrypting ballots with the public key of the Helios server before it leaves the voters device. Because the vote is encrypted, the voter needs a mechanism to verify that the correct vote was included. Cast-as-intended verifiability is provided by allowing the voter to choose between two actions after encrypting the ballot. Either send or audit the ballot. Auditing means decrypting the vote and exposing the included vote, thereby exposing a falsely included vote. The voter software does not know what the voter will choose. Thereby this mechanism should prevent malicious software from changing the voter's choice without her noticing it. After sending the ballot, it is stored on a central server encrypted and linked to the voter ID. Recorded-as-cast verifiability is provided because the voter can query the server for her ID and see if the encrypted ballot matches the one she sent. To ensure BP, the votes have to be separated from the voter IDs before decrypting them for the vote counting. Therefore, the ballots are shuffled in a mix-net without the voter IDs. The Helios server mixes the ballots and produces shuffle proofs showing that the mixing was done correctly. Observers of the process can verify the correctness and determine if the Helios server is honest or compromised. After shuffling, the server decrypts the ballots, produces a proof for correct decryption, and tallies the plain text votes.

In this original Helios protocol, BP heavily relies on trust in the Helios server. The server has all the cryptographic key material to decrypt ballots at any time. The shuffle and decryption proof only guarantee the integrity of the election result but do not help with privacy.

Later papers improved Helio's BP by replacing the simple ballot encryption with homomorphic encryption and distributed decryption [BCPS+11]. Homomorphic encryption removes the need for decrypting ballots. Instead, vote-tallying can be performed on the encrypted ballots, and only the result has to be decrypted. For this to improve BP, a set of trustees need to share the decryption key. A number of trustees over a specified threshold must work together to decrypt the election result. That implies that for decrypting ballots, the same number of trustees is required, which improves the trust assumptions for BP. Trust is now distributed.

Two other research projects that produced similar voting systems were conducted in Switzerland at École Polytechnique Fédérale de Lausanne (EPFL) [CGJ17] and the Bern University of Applied Sciences (BFH) [Hae13]. The system from EPFL exchanges the single server present in Helios with a multi-node setup based on their custom blockchain implementation called Skipchain [NKJG+17]. A single point of failure for storing ballots is thereby removed, and the system is naturally ready for other distributed mechanisms, like distributed key generation. The protocol otherwise follows the same procedure as Helios but applies a different shuffle algorithm.

[2]`https://heliosvoting.org/`

Developed at BFH, UniVote differentiates itself from the above two by additionally providing participation privacy towards the public. Although voters have to register with an identity, the election authorities shuffle those identities before ballots are cast, thereby anonymizing the voters after the registration. Similar to the ballots shuffle, the voting authorities produce a proof to attest the correctness of this shuffle as well [HS11].

All of the above systems rely on encryption for BP, and that encryption relies on computational intractability assumptions. Therefore, they do not provide everlasting BP, as described in Definition 6. If the intractability assumption becomes invalidated in the future, the ballots can be decrypted by whoever stored them. In the case of UniVote, this is only a problem if the parties that shuffle the voter identities remember the connection between voter identities and election credentials used to cast votes. In the other two systems, anyone has access to the information that links voters to their ballots.

The next section discusses approaches that introduce everlasting BP.

3.3 Protocols with Everlasting Privacy

The first protocol described here is an enhancement to Helios [DGA12]. The scheme adds a second channel between the voter client and the Helios server. In the original Helios implementation, voters send their ballot homomorphically encrypted to the Helios server where they publicly displayed. In this protocol, the encrypted ballot is not published but only available to the Helios server. Instead, a second message with a Pedersen commitment of the ballot is published. The client sends the encrypted random value used in the commitment, the encrypted ballot, and the commitment to the server. The server publishes only the commitment. The voter has to provide a zero-knowledge proof that she knows the opening of the commitment scheme and that the committed ballot is the same as the encrypted ballot. The Helios server verifies those proofs and accepts the ballot if they are valid. At the end of the voting period, the Helios server homomorphically tallies the ballots, and the trustees decrypt the result in a distributed manner. The server also generates and publishes proofs that show the counted ballots are congruent with the posted commitments.

The everlasting BP is based on the fact that an adversary does not have access to the encrypted ballots anymore. In case the underlying cryptosystem becomes broken in the future, there is nothing to gain for an outside observer. An attacker would have to intercept the private channel between a voter and the Helios server because the messages exchanged on this channel contain the encrypted ballot. If that channel's security is also based on computational intractability, the attacker might be able to recover the ballot, assuming that she recorded all communications.

In [CPP13], the authors seek to solve the same issues as the above scheme but with better performance. A novel encryption system is presented. The so-called consistent commitment encryption allows the voting authorities to derive commitments from encrypted ballots. Similar to the last protocol, only the commitments are publicly available, while the encrypted ballots are only visible to the voting authorities. Everlasting privacy is again provided towards the public but not towards the voting authorities. They receive the ballots directly from the voters and can remember that link until the underlying computational intractability is broken. Furthermore, a shared key is required to prevent a single authority to decrypt ballots at will.

The protocol described in [BDG13] combines the Helios approach with a mix-net removing the voter identities from the votes. The difference to a usual mix-net approach is that two separate mixes are performed simultaneously, a private one that shuffles the encrypted ballots and a public one that shuffles commitments to those ballots. This process can happen on the same machine. Only the public shuffle produces publicly verifiable information that proves the correctness of the shuffle.

Voters create a ballot, commit to it and encrypt it as well as the random value used in the commitment. The triple of the encrypted ballot, encrypted random value, and commitment is sent to the mix-net. The first mixer takes a batch of these triples and randomizes the commitments, re-encrypts the encrypted ballot, and homomorphically updates the encrypted random value. Then the triples are permuted and sent on to the next mixer. The last mixer publishes the triples. The ballot and the random value can now be decrypted. Even though the mix-net updated the commitment and the random value, the ballot and the random value are still a valid opening for the commitment. The mixes also need to provide a ZKP proving that the permutation they produced is a permutation of the input triples.

If the cryptosystem used for ballot encryption breaks in the future, only the first mixer can lift BP. It is the only party that has seen the encrypted ballots still linked to the voters. The authors propose a solution to this. The idea is to split the message from the voter into multiple parts and have multiple mix-nets handle each part separately. Thereby no mixer can get the full ballot information without conspiring with the other mixers. With this addition, the protocol introduces much complexity but also almost reaches unconditional BP. Only if all mixers collude, they can gain knowledge of the encrypted ballots and break privacy once the underlying cryptosystem is broken.

3.4 Towards Unconditional Privacy

The protocols in the last section introduced everlasting BP, making privacy independent of computational intractability. However, that privacy is only maintained towards the public, while for the voting authorities, the possibility remains to collude, thereby breaking BP. One might argue that for countering this, it is sufficient to distributing power among several authorities. Nonetheless, an effort is put into the construction of protocols that remove that last bit of trust into the authorities and try to achieve unconditional BP.

The first step towards unconditional BP is perfect BP. This notion of privacy ensures BP does not rely on trust in the voting authorities. One protocol with this property is proposed in [KY02] and is only meant to be used for small boardroom elections because it is data intensive. Every voter needs to generate and post data of size linear to the number of voters taking part in the election. The protocol is self-tallying, which means that anyone can do the vote counting. No voting authorities are necessary for the tallying process. The protocol requires a PBB where voters can post their ballots.

In [Gro04], a protocol is proposed that fulfills the same properties but is more efficient. Voters take turns in casting their ballots, each taking the current state of the election and modifying it by applying their ballot to it. The protocol offers anonymity when casting the ballot. However, for this mechanism to work, each voter has to use the public keys of all other voters, which limits the protocol's scalability.

Both of the above protocols depend on the DL assumption for their BP. Therefore, they do not provide everlasting BP and do not reach unconditional BP. In Section 4.1, a REV protocol is presented that fills this gap and makes the step to combine everlasting and perfect BP into unconditional BP.

3.5 Public Bulletin Boards for REV

PBBs appear as a fundamental component in some REV protocols [JMP13], and so there exists research focussing specifically on this concept. A short review of some work on PBBs is presented here.

In a voting protocol presented in [CGS97] the PBB appears as a distributed system, run by multiple servers. Access to the board is controlled via digital signatures, based on a public key infrastructure (PKI). There is no mention of how messages integrity is checked apart from the signatures attached to them. The participating servers run a byzantine agreement protocol to guarantee liveness of the board as long as at most one third of the servers are compromised. No detailed explanation of this process is given, but it is reminiscent of the consensus mechanisms used in some blockchains.

The work of [HL09] presents a possible PBB with only one central server. It is based on a hash chain of messages similar to a blockchain, though the concept of blocks is not used. Each message in the chain references the last message by its hash. When generating the hash of a message, it must include that reference, consequently inducing a completely interlinked chain. If one message in the chain is modified, the hashes of all subsequent messages are affected. When posting messages, the sender first fetches the latest message hash (called state hash) and a timestamp, both signed by the PBB server. The server cannot send the same state hash to multiple requesting senders simultaneously. Only one writer can use the current state hash to post his message. Once the sender's message is appended to the chain, the next writer can fetch the new state hash and prepare her message referencing the new state hash. The PBB server signs messages before appending them to the chain. Modification of the board's history is prevented as follows. When a reader queries a specific message, the board responds with the message, its hash, and the current timestamp signed with the server's private key. The reader can use this binding information at a later time to check if the PBB's contents have changed for any messages older than the received message. In conclusion, the proposed PBB makes sure that manipulations of the board's contents are detectable, but it does not prevent the manipulation in the first place. With only one PBB server, it is also not robust to failures.

The paper proposes an extension to transform the PBB into a distributed system. The problem now, of course, is that all participating servers need to have the same copy of the board and collaborate when appending new messages. The system requirements approach the ones of a blockchain.

In [HH16] an attempt for a formal and general specification of a PBB for REV is made. More precisely, the authors define methods that such a board should have and describe a few possible structures of the board and its messages. One possible board structure, which they call *interlinked*, is based on a hash chain similar to [HL09]. For controlling write access to the PBB, the paper assumes a known set of public keys connected to voters. The voters use the corresponding private keys to sign messages sent to the PBB. To give a consistency guarantee of the board's contents, the PBB signs every response to queries and also adds its own signature to messages that it adds to the board. Combined with a hash chain, these signatures represent commitments to the board contents at a particular time.

Finally, related to the append-only property of the PBB, a consideration of the right to be forgotten of the EU's General Data Protection Regulation (GDPR) is advisable when implementing a PBB. Since modifying or deleting the contents of the board is prohibited and the board contains personal data, *e.g.,* voter's ballots, the question arises if the board does break GDPR's right to erasure. As GDPR states in Art.4(1), personal data is data that can be related to an identifiable person. Identifiable means the person can be identified directly via a distinct identifier or indirectly via factors like psychological or economic attributes that reveal the person's identity [Eur16]. This is the case for voting systems that post direct voter identifiers or pseudonyms to the PBB. Though, such systems might evade the right to erasure by point (d) in Art. 17(3), which says that the right to erasure does not apply if the personal data is needed for achieving purposes in the public interest [Eur16]. In the case of a popular vote, which has to retain ballots for possible recounting, this paragraph seems to be applicable. In smaller voting setups performed in a private setting, the situation is not immediately clear.

Chapter 4

System Design

This chapter outlines the design of a voting system prototype, documenting the crucial steps form the protocol to its implementation. It starts with the introduction of the underlying voting protocol, documents the design of its most important component the PBB, discusses several design considerations, and proposes a system architecture.

4.1 A REV Protocol with Unconditional Privacy

This thesis focusses on the protocol presented in [LH15] and [Loc16]. To our knowledge, it is the first REV protocol that offers unconditional PB, *i.e.,* everlasting and perfect PB. This section introduces the protocol, its core principles, assumptions, and implications on an implementation of it. From here on, the protocol is referred to as the "protocol", the "voting protocol" or the "UP protocol" for Unconditional Privacy protocol.

4.1.1 The Protocol

The main participants in the protocol are voting authorities, voters, the PBB, and verifiers. It is comprised of four phases.

Registration Phase The voter creates private and public credentials and registers them. That is, she posts the public credential to the PBB, identifying herself with a known identity.

Preparation Phase The voting authorities check the eligibility of the registered voters, compile a list of eligible voters linked to their public credential and calculate the election polynomial. The list of voters and the polynomial are posted to the PBB.

Vote Casting Phase The voter checks for inclusion in the list of eligible voters, creates a ballot, and produces proofs that prove her eligibility without revealing her identity. She then posts the ballot to the PBB over an anonymous channel.

Tallying Phase Anyone can fetch the ballots and the polynomial from the PBB, verify the validity of the ballots, and calculate the election result.

Two multiplicative, cyclic groups in which the DDH assumption holds are required in the protocol. The first one $\mathbb{G}_p$ of prime order p and the second one $\mathbb{G}_q$ being a subgroup of $\mathbb{Z}_p^*$ with prime order q. As described in Section 2.2 Schnorr groups are a possible variant for such groups. In that case $\mathbb{G}_p$ is chosen to be a subgroup of $\mathbb{Z}_o^*$ where $o = pr + 1$ for some integer r with o and p being prime. $\mathbb{G}_q$ is chosen to be a subgroup of $\mathbb{Z}_p^*$ where $p = qk + 1$ for some integer k with q being prime. The modulus of $\mathbb{G}_p$ is o while the modulus of $\mathbb{G}_q$ is p, *i.e.,* the order of $\mathbb{G}_p$. Using Definition 22 we find generators $g_1, g_2 \in \mathbb{G}_p$ and $h_1, h_2, h_3 \in \mathbb{G}_q$. It is important that the generators are independent. That is, their relative discrete logarithms are unknown. At the example of g_1 and g_2 this means that $log_{g_1}(g_2)$ and $log_{g_2}(g_1)$ are not known. The Pedersen commitment scheme relies on this to fulfill the binding property.

In the **registration phase** (see Figure 4.1) the voter picks private credentials α, β at random from $\mathbb{Z}_q$ and computes a public credential $u = h_1^{\alpha} h_2^{\beta}$. $\mathbb{Z}_q$ is the set of integers modulo q. Note that by using the generators from $\mathbb{G}_q$, the application of the group operation - multiplication modulo q - is implied. Therefore, u is again an element of $\mathbb{G}_q$. The voter sends u to the PBB together with some identifying information. It is assumed that an Identity Management (IdM) system is available that allows the voting authorities to check a voter's eligibility.

After the registration phase is over, the voting authorities begin the **preparation phase** (see Figure 4.2). Based on the eligibility checks, the authorities publish a list $U = ((V_1, u_1), \ldots, (V_N, u_N))$ of eligible voters, where V_i is the voter's identity, u_i is the corresponding public credential, and N is the electorate size. From the set of public credentials, a polynomial is calculated.

$$P(X) = \prod_{i=1}^{N} X - u_i$$

$P(X)$ is defined as a polynomial in the polynomial ring $\mathbb{Z}_p[X]$, meaning that the coefficients are in $\mathbb{Z}_p$ and therefore operations on the coefficients are performed modulo p, *i.e.,* in the ring $\mathbb{Z}_p$. Note that $\mathbb{Z}_p$ is used and not $\mathbb{Z}_q$ even though $u_i \in \mathbb{G}_q$. The polynomial is needed for the proofs generated later in the protocol. Its coefficients $A = (a_0, \ldots, a_N)$ are posted to the PBB along with the list of voters U. Anyone can check for the correctness of the polynomial by recalculating it from U. Finally, the voting authorities choose an election generator $\hat{h} \in \mathbb{G}_q$ and post it to the PBB.

With the election polynomial and the election generator available on the PBB, the **vote casting phase** can begin (see Figure 4.3). The voter uses the election generator and part of his private credentials to generate an election-specific credential $\hat{h}^{\beta} = \hat{u} \in \mathbb{G}_q$, called election credential. Instead of identifying herself with the public credential u, the voter sends $\hat{u}$ with the vote. This removes the voter's identity from the vote, requiring three ZKPs to prove eligibility.

First, the a commitment $c = com_p(r, u)$ to the public credential is created, with $r \in_R \mathbb{Z}_p$. This is used together with the credential polynomial to create the first ZKP $\pi_1 =$

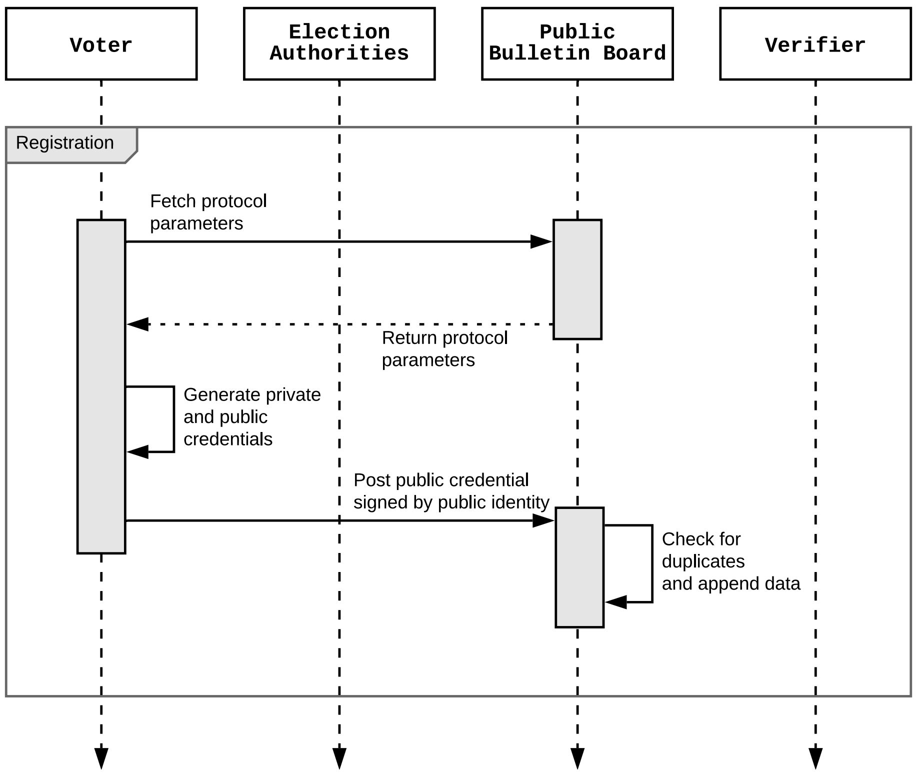

Voter
Election Authorities
Public Bulletin Board
Verifier
Registration
Fetch protocol parameters
Return protocol parameters
Generate private and public credentials
Post public credential signed by public identity
Check for duplicates and append data

Figure 4.1: The sequence diagram of the registration phase as envisaged by the UP protocol.

NIZKP$[(u, r) : c = com_p(r, u) \wedge P(u) = 0]$. This is a set membership proof, proving that c is a commitment to one of the credentials in the list of voters.

A second commitment $d = com_q(s, \alpha, \beta)$ to the private credentials is created, with $s \in_R \mathbb{Z}_q$. Commitments d and c are used to create the second proof $\pi_2 =$ NIZKP$[(u, r, s, \alpha, \beta) : c = com_p(r, u) \wedge d = com_q(s, \alpha, \beta) \wedge u = h_1^\alpha h_2^\beta]$ that proves knowledge of the private credentials α and β, which were used to generate u.

Finally, the voter needs to proof that it was the same β used to generate the election credential $\hat{u}$ as the public credential u. A proof of equality of discrete logarithm can be used for this and we denoted it as $\pi_3 =$ NIZKP$[(s, \alpha, \beta) : d = com_q(s, \alpha, \beta) \wedge \hat{u} = \hat{h}^\beta]$. It prevents the voter from sending multiple ballots with different $\hat{u}$.

The final ballot is the tuple $B = (v, \hat{u}, c, d, \pi_1, \pi_2, \pi_3)$, where v is the vote. The vote itself can be added to the ballot in plaintext. Therefore, it can be of any format. However, the voter cannot just add any vote to the ballot. Shee needs to make the ZKPs dependent on her vote. When creating the random challenge for the ZKPs the vote is part of the input into the hash function. This binds the vote to the ballot because it is again used as input when verifying the proofs. If another vote is sent with the ballot, the proof verification

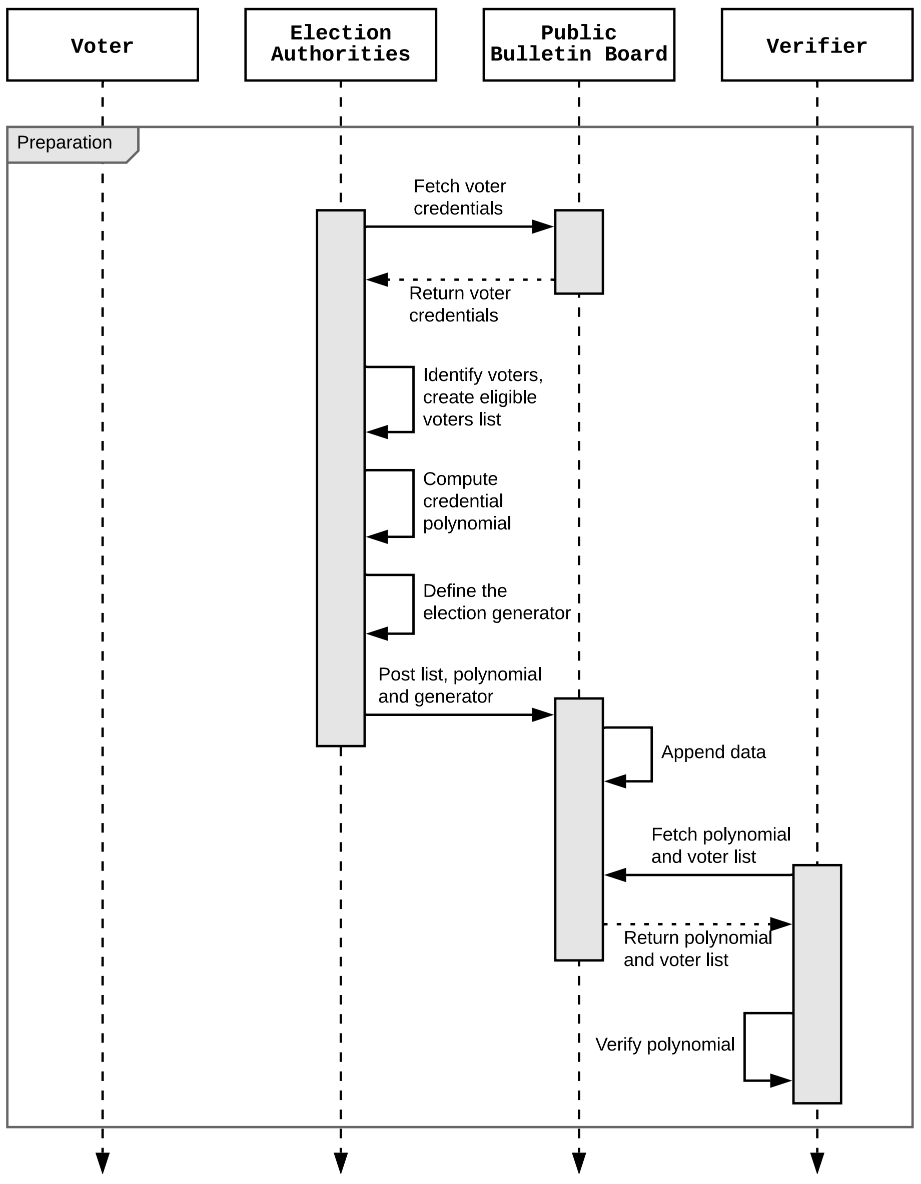

Voter
Election Authorities
Public Bulletin Board
Verifier
Preparation
Fetch voter credentials
Return voter credentials
Identify voters, create eligible voters list
Compute credential polynomial
Define the election generator
Post list, polynomial and generator
Append data
Fetch polynomial and voter list
Return polynomial and voter list
Verify polynomial

Figure 4.2: The sequence diagram of the preparation phase as envisaged by the UP protocol.

will fail.

The voter sends the ballot to the PBB using an anonymous channel that does not allow linking her ballot to her network address.

After the voting period is over, the **tallying phase** begins (see Figure 4.4). Anyone can fetch the credential polynomial and the ballots from the PBB, verify the proofs on each ballot, and accumulate the plain text votes from the valid ballots. If duplicate votes are

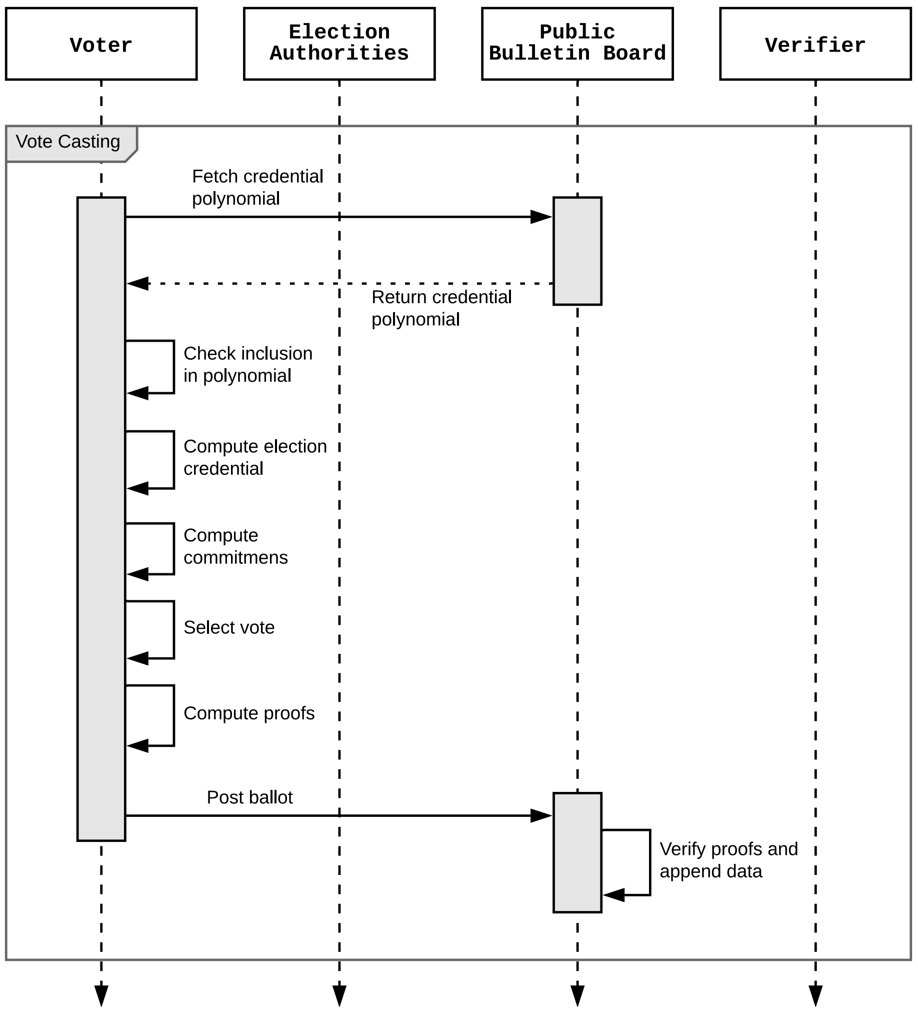

Voter
Election Authorities
Public Bulletin Board
Verifier
Vote Casting
Fetch credential polynomial
Return credential polynomial
Check inclusion in polynomial
Compute election credential
Compute commitmens
Select vote
Compute proofs
Post ballot
Verify proofs and append data

Figure 4.3: The sequence diagram of the vote casting phase as envisaged by the UP protocol.

allowed, they need to be cleaned at this point, *e.g.,* by simply dropping all but the last ballot corresponding to the same $\hat{u}$.

4.1.2 Threat Model

The threat model of the protocol divides adversaries into present and future adversaries. The present adversary is active at the time an election takes place, *e.g.,* by trying to produce a fake ballot. It is assumed that the present adversary is computationally bound. I.e., she cannot break assumptions about computation infeasibility, like the DL assumption. The future adversary, in contrast, is attributed with unlimited computational resources because, in the future, a cryptographic assumption used in the system might be broken,

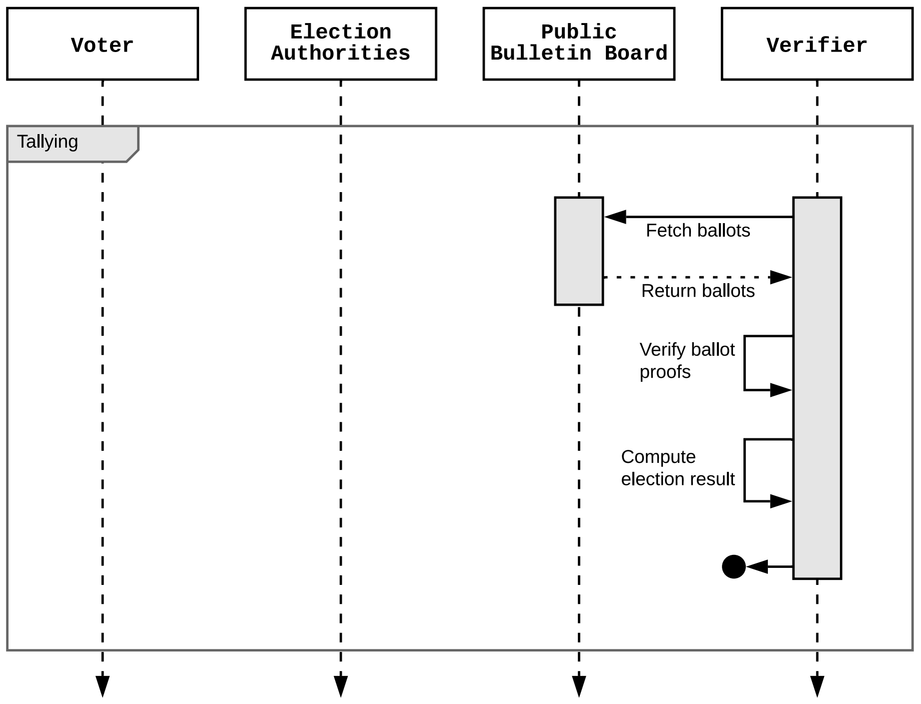


Figure 4.4: The sequence diagram of the tallying phase as envisaged by the UP protocol.

or the computational resources are far more powerful, making today's infeasible problems feasible. Not all parts of an e-voting system have to be secured against the future adversary. Constructing a valid but fake vote is of no use to her, since the election will be long over. Only BP matters in the context of a future adversary.

4.1.3 Properties of the Protocol

The UP protocol was chosen in this work because of its privacy properties. These and other voting-related properties are examined in this section.

Privacy

The protocol claims to provide unconditional BP. Thus, privacy does not depend on a trusted third party nor computational intractability. Even though the voter identifies herself in the registration phase, the only thing that is published in the voting phase are commitments but not the identifying information itself. The commitments to α, β and u are perfectly hiding due to the nature of Pedersen commitments described in Section 2.2.2. No information about α, β, and u can be gained from the commitments even if the DL assumption is lost in the future. The election credential $\hat{u} = \hat{h}^{\beta}$ will give away

β as soon as the DL assumption is gone. But, because $u = h_1^\alpha h_2^\beta$ is like a perfectly hiding commitment to β with randomness α, one cannot decide to which u the β belongs without also knowing α. Furthermore, all ZKPs are perfectly zero-knowledge. That is, no additional information can be gained from them apart from the fact that they prove.

To maintain privacy when sending the ballot to the PBB, an anonymous channel is required. Without such a channel, the receiver of the ballot (*e.g.,* a PBB server) can narrow down the possible originators of the vote by looking at the network address of the sender. Other entities in the network can try to identify the sender of a ballot as well, for example, by recording network activity of a host and comparing the timing of posted ballots with the host's connections to the PBB.

The protocol does not provide receipt-freeness. The voter can produce an opening for her commitment $c = com_p(r, u)$ by sharing the random value r to the corresponding public credential u. Thereby she can prove that she is indeed the originator of the ballot containing c. Reveal the private credentials is not necessary for producing the receipt.

The same reasoning applies to coercion-resistance. The protocol is not resistant to coercion. Though, if allowed by the PBB, a voter could repeatedly post ballots, overriding ballots that previously have been forced by a coercer.

Verifiability

The protocol provides individual verifiability by posting ballots on a PBB that has the properties described in Section 2.1.5. After casting a ballot, the voter can check via the PBB if the ballot was correctly received and recorded. However, this assumes that the voter's platform is not infected by malware corrupting the voter's view on the PBB. Malware on the voter's device can intercept all communications between the voter and the PBB, thereby modifying the voter's ballot and reflecting a wrong state of the PBB in order to keep the voter from noticing the modifications. Theoretically, individual verifiability can only be provided if we assume a trusted platform on the voter's side, but this is impractical. A more practical solution that gives some relief to the problem is to have the voter check the ballot with a second independent device.

Note that the voter should not query the PBB for her specific ballot over a non-anonymous channel. This would give away the link between voter and ballot to an observing PBB server. Instead, the voter should query a range, the whole set of ballots, or use an anonymous channel.

The protocol provides universal verifiability by the fact that anyone can calculate the election result. However, because ballots cannot be linked to specific voters, it is of great importance that eligibility is ensured. This is the purpose of the three proof transcripts π_1, π_2 and π_3 attached to the ballots. They prove that a voter is in the set of eligible voters, that she possesses the private credentials behind the public credential, and that the same private credentials were used to produce the election credential. To successfully cast a fake ballot, an attacker would have to find α and β for some $h_1^\alpha h_2^\beta = u \in U$ which should not be possible for a present time attacker because of the DL assumption. The other option is to construct a proof without knowing α and β. This is not possible for a present adversary because the proofs are computationally sound as described in Section 2.2.3.

Fairness

The protocol does not provide fairness by default. All votes are posted to the PBB in plaintext, allowing anyone to calculate intermediary election results at any time in the voting period. Fairness can be achieved by encrypting the vote with a public key provided by the voting authorities. The corresponding private key should be a shared secret amongst the voting authorities to distribute trust. Once the voting period is over, the parts of the private key can be published on the PBB, giving everyone the possibility to decrypt the votes. Note that the distributed trust needed for fairness does not affect unconditional ballot privacy.

4.2 Blockchain-based Public Bulletin Board

Most steps of the UP protocol rely on the existence of a PBB with the properties described in Section 2.1.5. Voters and observers should be able to write and read from the PBB directly. To uphold the trust assumptions of the protocol when transferring it into practice, the PBB should require minimal trust in its operators. A single trusted operator has the power to change election parameters and drop or invalidate ballots on the PBB. Therefore trust must be distributed either among a set of voting authorities or even third party operators. The work on PBB reviewed in Section 3.5 proposes the use of hash chains for storing ballots and a byzantine agreement scheme for communication between operators of the PBB. This hints at the applicability of blockchain. Combining the hash chain and the agreement scheme makes modifications of appended messages detectable, and at the same time, modifications are actively prevented.

Blockchain fulfills the requirements stated in Definition 15 of the PBB.

- The blockchain places messages in blocks and adds the blocks to the chain in sequence. The newest block contains the most recently posted messages.
- The cryptographically linked data structure of a blockchain allows the detection of modifications and deletions by anyone possessing a copy of the chain. The consensus mechanism prohibits modifications from happening in the first place.
- A blockchain is a distributed storage and, therefore, robust. E.g., against node failures.
- The consensus mechanism prevents invalid messages from being appended as long as a majority of participating nodes are honest and enforce the validation checks.

Since the release of Bitcoin, a multitude of different blockchain types have been proposed and implemented in practice [ZXDC+17]. Choosing the right blockchain platform for the PBB is the goal of the next few sections. Each section narrows down the selection by inspecting blockchain properties in the context of the PBB and the UP protocol.

4.2.1 Types of Blockchain

This section distinguishes between different blockchain types and argues which type fits the use case of the PBB best.

First, blockchains can be categorized with regards to their Read and Write access granted to network participants. Read access relates to the possibility of accessing the blockchain data itself (*e.g.,* transaction data). It can be divided into public and private. Write access refers to the possibility of participating in write operations to the blockchain, or participating in the consensus mechanism by becoming a *miner* (in PoW consensus mechanisms) or a *validator* (in most other consensus mechanisms). It can be divided into permissionless and permissioned. Note that this categorization is not standard by any means. Some authors might argue that some of these types do not even fall under the definition of a blockchain.

Public Permissionless Blockchain A public permissionless blockchain allows anyone to read and write and become part of the consensus mechanism. There is no central entity or consortium responsible for running the network. Taking part in the consensus and honest behavior is usually economically incentivized, making sure that participants play by the rules and are eager to keep the network alive.

Public Permissioned Blockchain In public permissioned blockchains, read access may still be granted to anyone, but write access and the right to participate in the consensus is restricted by an access control layer.

Private Permissionless Blockchain A private permissionless blockchain differs from public permissionless by restricting read access to a group or community. Thus, the write and read permissions are given to all participating members of this private group.

Private Permissioned Blockchain In private permissioned blockchains, read and write permissions are restricted. Hence read access, write access, and the power to participate in the consensus mechanism is only granted to select participants of a group.

We can further differentiate between dedicated and multi-purpose blockchains.

Dedicated Blockchain A dedicated blockchain is built for a single application. Bitcoin is an example of such a blockchain. Its primary application is value transfer[1]. However, dedicated can also mean that an organization deploys a custom blockchain network for application-specific purposes.

[1]Although, to some extend, implementation of other applications is possible via the Bitcoin virtual machine.

Multi-Purpose Blockchain A multi-purpose blockchain allows us to run multiple applications on the same network. Usually, this implies the use of smart contracts that enable the implementation of custom functionality on top of the basic blockchain functionality. Ethereum[2] is an example of such a blockchain.

A few examples of the above types are given here. Ethereum's main network is a public, permissionless, multi-purpose blockchain because it is open to anyone (read, write access, and participation in the consensus) and allows anyone to run their custom application via smart contracts. Hyperledger Fabric[3] is an example for a permissioned, multi-purpose blockchain. It's multi-purpose because a consortium can run different applications on the same network instance via the support of smart contracts. It is permissioned because it offers elaborate membership and access control features. It can be deployed in a private or public manner allowing or disallowing the public to read from the chain. The main network of Bitcoin is an example for a public, permissionless, dedicated blockchain. It is open to anyone but mostly aimed at value transfer. A permissioned, dedicated blockchain could be constructed from an application-specific implementation of Parity Substrate[4].

In the following, the blockchain types applicable to a PBB for the UP protocol are narrowed down. Public, permissionless, multi-purpose blockchains like Ethereum offer an already existing and running network. It is assumed that the consensus mechanism, and therefore trust, is distributed over a set of independent participants with an incentive to keep the network alive and only allow valid transactions to be included in the chain. Therefore, any application deployed on such a blockchain does not have to worry about setting up a network and running the consensus. Public, permissionless blockchains usually work with an account system in which every user has a public pseudonym and a corresponding key pair of which the private key is used to sign transactions. If used for the PBB, every voter would require such an account to interact with the blockchain. This poses a problem if used with the UP protocol. If voters have to sign the transaction that contains their ballot they leave a trail that could compromise their anonymity and, therefore, BP. Even though the blockchain accounts are not directly linked to a voter's identity, they can be used to attempt deanonymization. Another problem is that the services of public blockchains are usually not for free. A native cryptocurrency and other tokens are used to pay for usage of the network. The voting authorities have to pay for deploying the voting smart contract while each voter's blockchain account requires a sufficient balance to cover the cost of their transactions.

An estimation of the costs for storing ballots on the Ethereum chain is presented in the following. In [LH15] a size estimation of 466.5 MB is made for all ballots with an electorate size of 10'000 voters. The Ethereum yellow paper states that storing 32 bytes costs 20'000 GAS [Woo14]. Thus, storing 466.5 MB costs 291'562'500'000 GAS. At the time of writing, the average GAS price was 0.000000015214773 Ether[5] and an Ether cost around 150 USD[6], which results in approximately 4436 Ether equal to 665'400 USD.

[2]`https://ethereum.org/`
[3]`https://www.hyperledger.org/projects/fabric`
[4]`https://www.parity.io/substrate/`
[5]Taken from `https://etherscan.io/`
[6]Taken from `https://coinmarketcap.com/`

Paying more than half a million USD only for storing the ballots of one election with 10'000 voters seems unreasonable. The costs for running other operations in the smart contract are not considered here, but since the UP protocol is computationally intense, they can be expected to be high. A solution to reduce the amount paid for storing ballots is to only store hashes of the ballots on the blockchain and keeping the actual ballots on a cheaper data storage, *i.e.,* centralized storage with public access but no append-only guarantees. However, this leads to different trust assumptions and makes the setup more complex.

We conclude that even though public, permissionless, multi-purpose blockchains have good trust assumptions, they are not the best fit for a PBB for the UP protocol. Anonymity and costs are problems for which solutions would introduce much more complexity to the voting system. Furthermore, it is unclear if the cryptography used in the UP protocol can be implemented with the smart contract languages offered by such blockchains.

To avoid cost issues, it makes sense to use a permissioned blockchain that is under the control of a restricted set of operators. For example, if a government already uses a permissioned, multi-purpose blockchain for other official and administrative tasks, a PBB could be integrated into it. On the other hand, if such infrastructure is not available, a dedicated, permissioned PBB blockchain makes sense. In both cases, the incentive to participate as a consensus node is not an economical one but rather one of preventing fraud and guaranteeing a smooth voting process. Therefore, it seems more reasonable to have an official and verifiable list of consensus participants instead of opening the consensus up to anyone. Requiring participants to be identifiable, pressures them to behave honestly. If anyone could take part in a consensus without economic rewards, malicious behavior is bound to occur. Trust in a permissioned blockchain is still distributed. No single participant can make the PBB stop or create invalid blocks. The necessary degree of distribution depends on the type of election. Though, it is not necessary and also not feasible that every voter takes an active part in the network.

The above argumentation concludes that a permissioned blockchain fits the requirements of a PBB best. Permission is needed to take part in the consensus mechanism and for some write actions. E.g., only the voting authorities should be able to post protocol parameters, and only voters that are registered in a separate IdM system should be able to register for an election. Thus, write access is restricted in the registration, preparation, and tallying phase of the UP protocol, but not in the vote casting phase. In the latter, the voters need to remain anonymous. Only the ZKPs can be used to filter out invalid ballots. Read access must be granted to everyone for verifiability, making it a public, permissioned blockchain. If the blockchain should be dedicated or multi-purpose is decided in Section 4.2.3 where specific blockchain platforms are presented.

4.2.2 Consensus Mechanism

The consensus mechanism of the PBB blockchain makes sure that multiple entities need to agree on what is valid and gets appended to the PBB. It prevents faulty or malicious actions compromising the PBB's contents. The consensus mechanism should not only be fault tolerant but also support the property of Byzantine Fault Tolerance (BFT). If the latter is given, the PBB is robust against a certain number of dishonest consensus nodes and can still safely continue in their presence. In the following several aspects of the consensus mechanism are considered in the context of the PBB.

Block Time

The block time is the time interval between the production of new blocks. Preferably, the production of blocks should run as follows. If no transactions arrive, no blocks are created. If the transaction inflow is small, blocks should be created in fixed time intervals to guarantee that the voters can see their ballots on the PBB soon after posting them. Thus, even if the maximum block size is not reached, a new block should be produced after a fixed time period. If the transaction inflow fills blocks up to their maximum size faster before the configure block time runs out, then blocks should be created as soon as they are full. Because such a dynamic block production is not often seen in blockchains, in the context of a PBB, it is most important that blocks get generated in reasonable time intervals, so that posted ballots are visible on the PBB as soon as possible.

Block Finality

Not every consensus mechanism ensures that a block added to the chain will stay on the chain. That property is called block finality. A consensus mechanism has instant block finality if it assures that every block added to the chain will remain on it. Such a mechanism is preferable for the PBB because of individual verifiability. The voter can be reassured that her ballot will remain on the PBB. A probabilistic block finality implies that a voter cannot immediately tell if her ballot was successfully registered. This introduces unnecessary uncertainty.

Incentive for Honest Behavior

As discussed before, the safety and liveness of the consensus cannot depend on economic incentives. The PBB of an election is not a platform that stores economic value, and there will be no cryptocurrencies along with the PBB that could be used for rewarding participants. The consensus mechanism must, therefore, rely on a group of selected nodes that have a reputation to lose and are interested in a functioning election system.

Network Size

The PBB does not need to be backed by a big network of independent nodes like it is the case with Ethereum. It should be able to run with only a small set of consensus nodes and some nodes maintaining a copy of the blockchain. However, the smaller the set of consensus nodes the less malicious nodes are needed to jeopardize the liveness or safety of the blockchain. Therefore, the consensus participants should be publicly known and observed while the election is running.

Summary

The above observations about the consensus mechanism can be summarised in the following points.

- Block finality should be instant.
- Block production should be dynamic or at least configurable to a fixed time interval.
- Only publicly known and authenticated participants should be part of the consensus mechanism.
- Honest participation in the consensus does not depend on economic incentives.

There is another aspect that needs consideration in the context of the consensus mechanism that is specific to the UP protocol. With the estimates given in [LH15], the ballot size can be expected to be about 50 KB for an electorate size of 10'000 voters. If a thousand voters cast a ballot in rapid succession, the blockchain network has to deal with 50 MB of transaction data. This can pose a performance bottleneck on the networking side of the consensus mechanism if the consensus mechanism requires a lot of gossiping between nodes.

4.2.3 Available Blockchains

Next to determining the blockchain properties required by the PBB, research was conducted on existing blockchain platforms, of which some are mentioned in this section.

Neo

The Neo blockchain is based on the so-called delegated Byzantine Fault Tolerant (dBFT) consensus mechanism [Neo], which is a PoS variation. Participants can vote for registered consensus nodes with their NEO tokens. With enough votes, a node can become a validator and take part in the consensus. As the name points out, the consensus has BFT. The active consensus nodes are rewarded with GAS, another token on the Neo blockchain. Although normal users get rewards for merely holding NEO, they do not get rewarded when their delegates generate blocks. Nodes participating in the consensus are identified with a public key and information about the organization that runs them. At the time of writing, the process of registering a new consensus node on the main Neo network takes quite some effort and requires an off-chain application to be sent to the Neo Foundation.

The main Neo network is not an option for our voting system because of the economic issues discussed in Section 4.2.1. Nevertheless, if deployed in a custom network, the dBFT consensus mechanism fits the PBB's requirements. It allows us to have a fixed and publicly known set of consensus nodes, has an instant block finality, and configurable block time. However, since Neo is a multi-purpose smart contract chain, the PBB would have to be implemented via smart contracts. Neo allows the use of several general programming languages (*e.g.,* Java, C#) for smart contract implementation but only in a restricted subset, which might prevent the implementation of sophisticated cryptographic features.

Ethereum

Ethereum originally is a PoW blockchain but has been extended with other consensus mechanisms that are less resource intensive. The PoA algorithm called Clique is one of them [Szi17]. It is a simple mechanism in which leaders and co-leaders take turns in proposing new blocks. As discussed in Section 4.2.1, the main Ethereum network is not an option for the PBB, but a custom deployment using Clique can still be. However, Clique is not BFT. Similarly to Neo, Ethereum is a multi-purpose smart contract blockchain. Thus, implementation of the cryptographical functionality needed for the UP protocol would have to happen in Ethereum's smart contract language Solidity, which might be an issue.

Parity Substrate

Parity Substrate is a framework for building custom dedicated or multi-purpose blockchains. Its goal is to provide a very modular architecture that allows, but does not necessarily require, customization on different abstraction layers. Different pluggable consensus mechanisms are available, although, from the project's documentation, it is not immediately apparent if one of them fits the requirements of the PBB. Only the mechanism called GRANDPA has finality under BFT, but it does not produce blocks by itself and needs to be combined with one of the other offered mechanisms [Par].

Substrate is implemented in the Rust[7] programming language. Thus, building an application-specific blockchain on it requires knowledge of Rust, which is a language with a steep learning curve. Furthermore, at the time of writing, the development state of Substrate did not seem mature enough to make it easily usable for a prototype implementation.

Tendermint and Cosmos-SDK

Tendermint, at its core, is a blockchain-based consensus engine enabling the implementation of a custom replicated state machine, or simply put an application, on top of it [Ten]. That is, Tendermint can be used to implement a custom dedicated blockchain. Its consensus mechanism is BFT with instant finality. Tendermint itself makes no assumptions about the application built on it and thus supplies a very general interface. It is extended by Cosmos-SDK that adds more abstractions, making the implementation of a custom application faster and more comfortable. Cosmos-SDK also provides several ready-made modules that extend Tendermint's basic functionality. For example, a staking system and

[7]`https://www.rust-lang.org/`

a PoS consensus mechanism running on top of Tendermint's consensus engine. A module for a PoA consensus mechanism is in the making[8].

Tendermint and Cosmos-SDK are written in Go[9]. While Tendermint allows writing applications in other languages, Cosmos-SDK requires implementations exclusively in Go. Go is a programming language that was designed with simplicity in mind and could, therefore, be a good choice for a prototypical implementation.

Hyperledger

Hyperledger is a community hosting multiple blockchain projects of which Fabric is a prominent and well established one [Lin]. Fabric is a modular, permissioned blockchain which allows to plugin different consensus mechanisms. A BFT consensus implementation called BFT-SMaRt [SBV17] exists, but according to the documentation of the most recent Fabric version 2.1, a BFT consensus mechanism is not part of Fabric [10]. Only fault tolerant mechanisms are supported.

Fabric is different from most other blockchains in how transactions are processed. Transactions require endorsements according to some endorsement policy to get accepted by the network. Users, therefore, have to first send transactions to endorser nodes that validate them. Only if the user collects enough endorsements, they can proceed with sending their transaction to the ordering service where the consensus mechanism takes place and blocks are produced. The advantage of this approach is that consensus nodes do not need to validate and run the entire transaction content, but only need to check for sufficient endorsements. This concept could ease the problem of denial of service attacks, specifically in the case of the UP protocol. The consensus is not immediately affected if the system is bombarded with a large number of ballots. The load is placed on the endorsers while the consensus nodes can continue producing blocks at regular intervals. Clients are forced to collect endorsements before they can send the transaction in a valid state to the consensus nodes.

Fabric is a multi-purpose blockchain in that it provides smart contract capabilities via so-called chaincodes. Chaincode can be written in Java, Go, or JavaScript without restrictions on the language. The only thing that the developer has to make sure is that the code runs deterministically. Otherwise, different endorsers will come to different results when running the code, and transactions will be rejected.

Fabric provides an elaborate membership service and separation of groups and applications via so-called channels. The membership service is of limited use to the UP protocolbecause the blockchain should not identify voters when they cast ballots.

[8]`https://github.com/cosmos/modules/pull/5`

[9]`https://golang.org/`

[10]`https://hyperledger-fabric.readthedocs.io/en/release-2.1/orderer/ordering_service.html#ordering-service-implementations`

4.2.4 Final Selection

The following requirements to a blockchain are an aggregation of the discussion in the last few sections. A blockchain should fulfill these to apply to the UP protocol's PBB.

- Write access to the PBB can be restricted via an identity system in the registration, preparation, and tallying phase.
- It is possible to allow unrestricted write access in the vote casting phase. Anyone should be able to cast a ballot.
- Read access to all data on the PBB can be permitted to everyone.
- The participants of the consensus mechanism are fixed and publicly known.
- The consensus mechanism has instant finality, fixed block time, and does not depend on economic incentives.
- The blockchain provides a simple but powerful enough interface for implementing the UP protocol's functionality, *e.g.,* the cryptographic proof verification.

First of all, Neo, Ethereum, and other similar blockchains, even if deployed in a permissioned network, are not an option because of the restrictions that their smart contract languages incur.

Hyperledger Fabric has an exceptional mechanism to handle incoming transactions that could prove useful for computationally intense transaction verification. Its chaincodes promise the convenient implementation of application-specific functionality in general-purpose programming languages. However, as far as we know, it incurs a considerable overhead for setting up and configuring a Fabric network with membership and access control features that are barely needed in the UP protocol. Additionally, the newest version of Fabric seems to lack a BFT consensus mechanism and only provides fault tolerant mechanisms.

Parity Substrate shows high aspirations by providing a comprehensive framework with much flexibility but also abstractions. At the time of writing, its development state seemed not mature enough, and the usage of Rust is daunting to a beginner. Additionally, setting up a fitting consensus mechanism did seem problematic.

Tendermint and Cosmos-SDK provide a balance of simplicity and functionality scope. The consensus mechanism is BFT with instant finality, and the block time is configurable. Even though the PoA consensus mechanism is still in development, the PBB can make use of Cosmos-SDK's PoS module without the need for the staking token having any value. Validator nodes can be set up in a fixed manner, even in the PoS module. Access control is possible via signatures on transactions, while Tendermint does not make any more assumptions about the identities behind those signatures. A separate IdM system can be integrated to provide keys for signing transactions. For ballot casting, voters can sign their transactions with a key pair that is public and used by every voter. That way, their anonymity is preserved. The use of Go and Cosmos-SDK's intuitive modular structure promises a flat learning curve even for beginners of the language. For all the above arguments Tendermint and Cosmos-SDK were chosen for the PBB implementation.

4.3 Design Considerations

Transferring the UP protocol into practice requires the elaboration of aspects that are not considered in its theoretical definition. This section discusses these aspects and proposes solutions to issues that pop up in the practical implementation of the protocol.

4.3.1 Proof Verification

A problematic aspect of the UP protocol is the generation and verification of the ZKPs contained in the ballots. The computationally intensive ones are the set membership proof π_1 and the proof of known representation of committed value π_2. The proof of known equality of discrete logarithms π_3 is neglectable in comparison. The effort spent on generation and evaluation of π_1 and π_2 depends on the choice of groups $\mathbb{G}_p$ and $\mathbb{G}_q$, *i.e.,* the size of their order and modulus. Furthermore, the time complexity for π_1 grows linearly with the size of the electorate, while π_2 depends linearly on an additional security parameter κ. Proof generation is the lesser problem because every voter only needs to produce one ballot at a time. Proof verification however, can become a significant problem because many ballots might need to be verified at once. According to the estimates in [LH15], verifying a ballot's proofs takes about one second with an electorate size of 10'000 voters.

There are two options for the timing of the proof verification. Either *(1)* proofs are verified at the time the ballot reaches the PBB or *(2)* the PBB stores all incoming ballots without checking the proofs and verification is only done later when the vote casting phase is over. Both options have their disadvantages. In option *(1)*, the computational intensity of the proof verification puts a heavy load on the blockchain nodes and the consensus mechanism. We anticipate that Denial of service (DoS) attacks will be easy to perform. Option *(2)* allows anyone to post any number of invalid ballots without rejection as long as the transaction abides the required format. The PBB can quickly be cluttered with a vast amount of invalid ballots. There is simply no other mechanism besides the actual proof verification that can discern an invalid from a valid ballot. In such a scenario, the verification work that has to be done in the tallying phase could become impractically high. Our implementation chooses option *(1)* because option *(2)* hands off any possible control till the end of the vote casting phase.

Partially verifying a ballot to speed up performance in the vote casting period is not an option. Proofs π_1 and π_2 are both essential in checking if a ballot comes from an eligible voter and if the sender of the ballot possesses the private credentials corresponding to that voter's public credential. Only verifying π_1 enables an attacker to post ballots with membership proofs for any public credential $u \in U$ without owning the corresponding private credentials. Only verifying π_2 does not tell if the public credential involved in the proof represents an actual eligible voter. Verification of the third proof π_3 could be delayed to the tallying phase because it only makes sure that an eligible voter does not cast multiple ballots under different election credentials $\hat{u}$. However, the impact of π_3 is small compared to the other two that it does not influence the overall performance of the verification process much.

To keep invalid ballots from reaching consensus nodes in the first place, other non-consensus nodes taking part in the network should always run the proof verification on ballots before further broadcasting them through the network. If a ballot is found to be invalid, it can be dropped immediately at any node, thereby not bothering the validators. On the other hand, verification on a valid ballot has to happen at every node that receives it. Relying on the approval of only one node is not acceptable in a decentralized blockchain setting.

4.3.2 Casting Multiple Ballots

The UP protocol in principal allows voters to cast multiple ballots that are resolved either immediately or later in the tallying phase. Our implementation does not allow this. The reasion being the prevention of replay attacks. We can save computations by filter out replayed ballots by checking if the election credential on the ballot was already used. This way, the PBB does not have to run through the whole proof verification. It is guaranteed that the eligible voter will be the first one to use the election credential $\hat{u} = \hat{h}^{\beta}$, because only she knows the private credential β. Only after a ballot with that $\hat{u}$ has been posted will an attacker be able to replay ballots with the same $\hat{u}$.

4.3.3 Identity Management

A peculiarity of the UP protocol is the anonymity of voters when posting ballots. All work on PBB presented in Section 3.5 proposes to use digital signatures on voter's messages, which is an intuitive way for a voter to prove her identity and thus her eligibility for casting a vote. However, in the case of the UP protocol, ballots are cast without revealing the voter's identity. Letting the voter add a signature to the ballot, which includes the vote in plain text, obviously removes BP and defeats the fundamentals of the protocol. On the other hand, when a voter registers to take part in an election, an identity known to the voting authorities is required. In our prototype, a voter identity, *i.e.,* a key pair, has to be created ad hoc on the voter's machine. The public key of the pair would then have to be registered in an IdM system, presumably controlled by the voting authorities, to which the PBB has access. When the voter registers for an election, she signs the registration transaction with the private key, and the PBB can check if the signature belongs to one of the eligible voters in the IdM system. If a match is found, the public credential u in the registration transaction is stored in the list of eligible voters. Note that the public credential u used in the UP protocol is not the same as the public key used to sign the transaction. Our implementation does not yet include an IdM system. Thus, the PBB cannot perform eligibility checks when processing registration transactions. I.e., anyone can register any number of new public credentials on the PBB. If an IdM system is added in the future, it will serve the blockchain as an oracle. An incoming registration transaction will be checked for its signature and the corresponding public key. The blockchain node will then call the IdM system and check if a voter identity is registered under the public key. If that is the case, the election registration is valid, and the public key, as well as the public credential u, is stored on the PBB in the list of eligible voters U. The public key can be used by anyone to verify that the credential u belongs to a voter registered in the IdM system.

The prototype does not yet provide any support for handling key pairs and election credentials. The public and private credentials u, α, and β that the voter needs to generate are stored on the voter machine's file system. In a real-world scenario, the voter cannot be burdened with key management, and a solution is required for this aspect of the system. Though, such a solution must not downgrade the trust assumptions of the overall system, *e.g.,* by trusting a single entity with storing the voter's private credentials.

Finally, the voting authorities require an identity for posting information on the PBB as well. The transactions with which the authorities post the election parameters, voter list, credential polynomial, and election results to the PBB have to be signed. The PBB should be configured such that it only allows the authorities to be the originator of this information.

4.3.4 Anonymous Communication Channel

The UP protocol's unconditional ballot privacy property relies on the existence of an anonymous channel between the voter client and the PBB. Without such a channel, it is possible to track down the originator of a ballot and link the plaintext vote to a specific voter. This section discusses possible approaches for providing voter anonymity when interacting with the PBB.

In the early days of Bitcoin, backers of the technology argued that one of its advantages is the privacy that it provides for its users. But over the years, research looking into that privacy property has shown that this assumption is not correct. Profiles of users of the cryptocurrency can be constructed even if the users take recommended privacy precautions [AKRS+13]. Several approaches have been developed to improve privacy, specifically for the Bitcoin network. One example is TumbleBit, which uses an off-chain mechanism to break the link between payer and payee, thereby allowing the users to remain anonymous over multiple payments [HABS+16].

However, this kind of anonymity-enhancing technology does not consider network-level information that an attacker can use to lift the anonymity of a blockchain user. Accessing the blockchain network to place a transaction and then retrieving information about that transaction leaves traces from the user. The same problem applies to the PBB. Even if the ballot itself does not reveal any information about the voter, the network metadata recorded when posting the ballot can reveal the voter's identity. Even privacy-focused blockchains, like Zcash[11], do not directly tackle this problem but usually refer to the application of an external anonymity service like the Tor project[12] [HHA18]. In [RMK17], a P2P mixing protocol is proposed that provides anonymity on the network access level. It is based on the idea of the Dining Cryptographers Network (DC-Net) [Cha88]. Multiple voters need to be active at the same time so that a DC-Net can be established among

[11]`https://z.cash/`
[12]`https://www.torproject.org/`

them. Additionally, the participants in a DC-Net need to know the public keys of all other participants. Making the availability of other voters a prerequisite for casting a vote is not practical in a voting system. Alternatively, establishing a DC-Net instance between individual voters and several voting authority nodes does not provide anonymity because it is evident that any ballot sent in such a network is sent by the voter.

Not only does casting a ballot require anonymity but also looking up the posted ballot. Checking if the ballot was recorded correctly on the PBB is the only way to provide individual verifiability in the UP protocol. This problem is studied in Private Information Retrieval (PIR) protocols, which allow a user to retrieve data from a database without revealing to the provider what data was retrieved [HHA18]. The simplest but also most inefficient of such protocols downloads the entire ballot list and checks for the existence of a specific ballot offline. This provides information-theoretic privacy, but in case of the UP protocol demanding from each voter to download all ballots becomes impractical for large electorates. Already for an electorate fo 10'000 voters, the accumulated ballot size is expected to be around 500 MB [LH15]. Confining the query to a subset of the PBB's ballots eases the data volume problem but also lessens the anonymity property. According to [HHA18], more elaborate PIR protocols are too inefficient to be of use in open blockchain systems, and more research effort is needed before PIR can be used for anonymous information retrieval from blockchains.

For the lack of other practical solutions, the intuitive choice providing an anonymous channel for posting and fetching ballots is an anonymity network like Tor. Although at the example of Bitcoin, it has been shown that using Tor to access a blockchain network can lead to worse privacy properties than without it [BP15]. However, these attacks do not directly apply to the setting of our PBB. In principle, they try to separate the users that connect to the blockchain via Tor form the rest of the network by forcing connections through the attacker's blockchain nodes. In the case of the UP protocol this attack does not provide any information about the voter because, even if her ballot reaches the PBB via an attacker's blockchain node, she is still anonymous. Other deanonymization attacks not specific to our blockchain case are possible, *e.g.,* the ones described in [BMGK+07].

Additionally, in theory, the usage of Tor changes the privacy properties of the UP protocol. Tor's anonymity is based on asymmetric cryptography, which in turn is based on computational intractability assumptions. A future, computationally unbounded attacker that has recorded all traffic going in and out of the Tor network at the time of an election can attempt deanonymization by encrypting all traffic. Practically, this attack is neglectable. Therefore, Tor remains a practical solution for providing anonymity in the UP protocol.

The anonymous channel is not part of the prototype implemented in this work. Although, an anonymous connection to the PBB via the Tor network can be established separately by forwarding all traffic from and to the voter client through a Tor proxy running on the client's machine.

4.3.5 Malicious Network Participants

Even though the PBB runs safely as long as at least two thirds of all consensus nodes are honest, individual nodes can still cause disruptions. A PBB node can censor incoming transactions, thereby inhibiting access to the election for specific voters. This is of little use in the vote casting phase, since voters are not identifiable through their ballots and can, therefore, not be targeted individually. However, in the registration phase, a node can effectively block a specific voter's attempt to register by dropping transactions received from that voter. For simple mitigation of this attack, the voter client should check the PBB after issuing a registration transaction and retry with another node if the registration did not work.

A variation of censorship is providing false information. A PBB node can, for example, respond with false protocol parameters to a voter's requests. In consequence, the voter will create her credentials based on false groups and generators. In the best case, these credentials will be rejected when posted to the PBB and, in the worst case, might go unnoticed if the public credential is also an element of the correct group. In the latter case, the voter will think she registered successfully but will not be able to generate a valid ballot in the vote casting phase, excluding her from the election. For detecting such fraud early on, the voter client should consult at least two PBB nodes when querying the protocol parameters. Less damage is done if a node sends an invalid credential polynomial. That will lead to an invalid ballot, to which the voter can react by fetching from another node and rebuilding her ballot.

4.3.6 Insecure Voter Platform

As in any REV system that allows voters to use their personal devices, the UP protocol has to deal with untrusted voter platforms. Malware compromising the voter client can (1) block the voter's ballot from being posted to the PBB, (2) choose to cast another vote in the name of the voter or (3) report the voter's choice to the malware originator. While attack (1) and (3) are possible with malware that exists outside of the voter client application, attack (2) assume that the attacker successfully modified the voter client executable itself. Attack (1) and (2) can both be detected by checking the PBB. However, for this check to be effective, the voter must use a second device to connect to the PBB because the malware can easily display fake content. For attack (3) no countermeasures exist in the UP protocol version used in this work. Once a client device is infected, the attacker can link any ballot produced with that device to the owner. The protocol extensions presented in [Loc16] might mitigate the problem, but is not part of this work.

The measures that can be taken to prevent infection of the voter client application depend on the way the application is distributed to the voters. Applications that are installed on voter devices can usually be protected via platform-specific app stores. Such app stores promise that software available on them is signed by the issuer and gives the user some confidence in the integrity of the voter client. If the client is distributed as a web application, the application code can be manipulated somewhere on the communication channel or in an infected browser or browser extensions. A possible countermeasure is the usage of the Subresource Integrity[13] feature provided by some modern web browsers. It allows the browser to check if the received application code is congruent with the expected code.

The minimal trust assumptions of the UP protocol make deployment of voter client applications more complicated. If the voters accept an application uploaded by the voting authorities, they should not blindly trust the integrity of that software, even if it was built from an open-source code base. Verification mechanisms need to be set up that enable anyone to build the voter client and compare the binary to the application distributed by the authorities.

4.3.7 Protocol Parameters

The UP protocol depends on the definition of the two groups $\mathbb{G}_p$, $\mathbb{G}_q$, and their generators. These have to be set up and posted to the PBB before the registration phase. This can be done by the voting authorities or other entities as long as it is guaranteed that the relative discrete logarithms of the generators are not known to anyone, *i.e.,* the generators are independent. If they are not independent, the Pedersen commitments lose their binding property. For example, knowing the relative discrete logarithm $x = log_{g_1}(g_2)$, one can come up with an alternative opening (α', β') for commitment $com_p(\alpha, \beta) = g_1^{\alpha} g_2^{\beta}$ by replacing g_2 with g_1^x, giving $com_p(\alpha, \beta) = g_1^{\alpha}(g_1^x)^{\beta} = g_1^{\alpha} g_1^{x\beta} = g_1^{\alpha+x\beta}$ and then finding $\alpha' + x\beta' = \alpha + x\beta$. A voter with this capability can cast multiple valid votes with only one registered public credential because she can generate multiple valid private credentials corresponding to that public credential.

For the protocol not to require trust in any single party, the generators have to be instantiated in a trustless or at least distributed manner. A possible solution for this is to have a common reference string (CRS) from which the generators can be derived. However, producing a CRS without trust in one entity introduces more complexity, as can be seen at the example of a multi-party protocol that was designed for the setup of a CRS for the Zcash blockchain [BGG19]. Nonetheless, such a protocol might be necessary for the UP protocol to achieve its privacy properties fully. Setting up the group parameters in an untrusted manner is not part of the prototype implemented in this work. The parameters are defined at the beginning of the election by whoever sets up the PBB network.

4.3.8 GDPR

As discussed in Section 3.5, the GDPR could pose a legal problem for blockchain-based systems. Applicability of GDPR to our voting system depends on the use case. If the system is used in public elections, point (d) in Art. 17(3) could apply, which means that the voter does not have the right to erasure because the data is needed for achieving purposes in the public interest [Eur16]. In private use cases, the matter is less clear.

[13]`https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity`

Anyhow, the only data that is connected to the voter is her identity and public credential registered in the protocol's registration phase. Due to the nature of the UP protocol the PBB does not store identifying information with the ballots. According to Recital 26[14], GDPR does not apply to such anonymous data as the ballots.

4.4 System Architecture

This section introduces the architecture of the voting system implemented in this work. Each component's purpose and interaction with other components, as well as specific issues of the component are described. Figure 4.5 shows the complete system architecture.

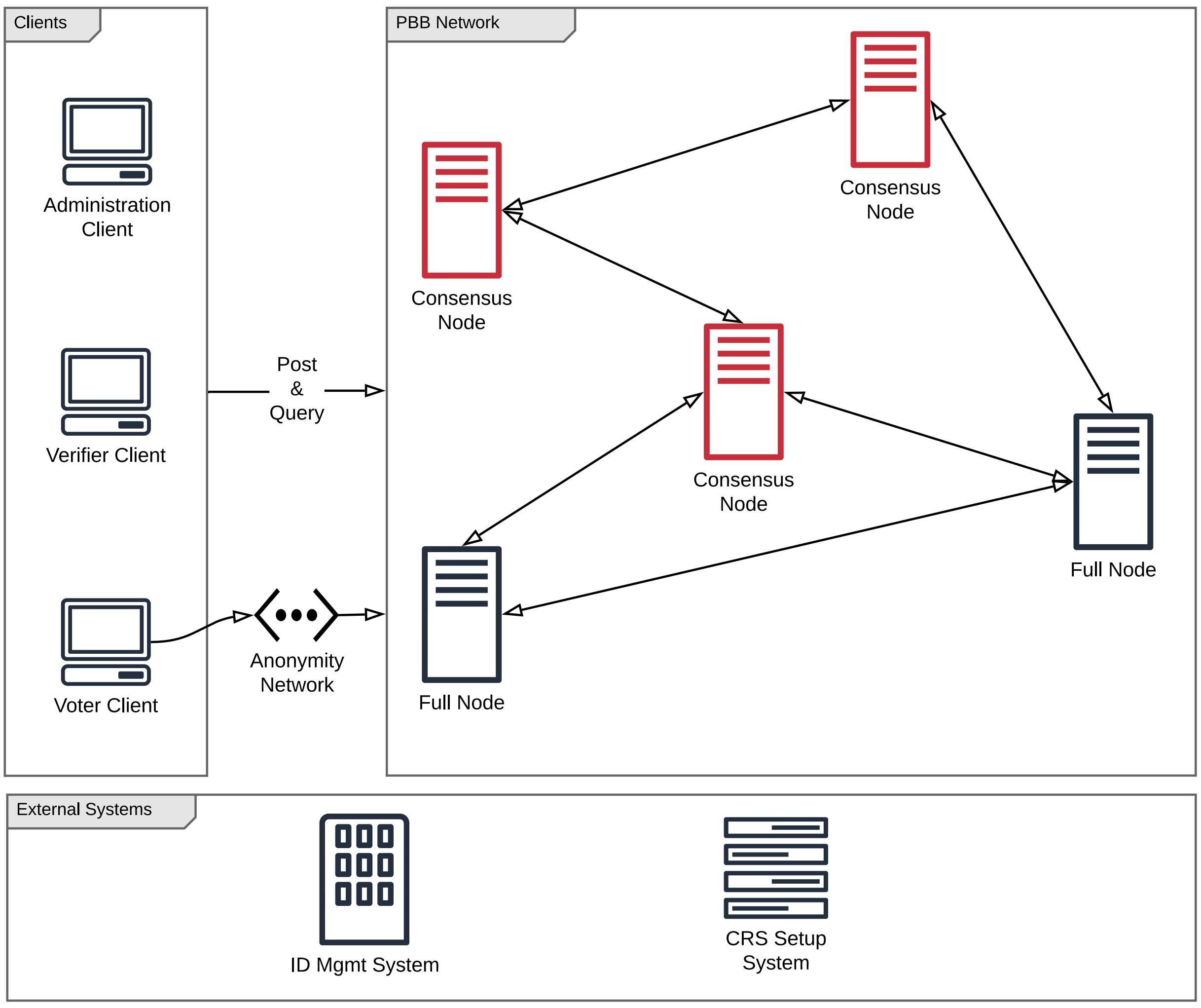


Figure 4.5: System architecture diagram of the voting system prototype.

Blockchain Nodes

The PBB is composed of a network of two types of nodes. The consensus nodes (red in Figure 4.5) are the main drivers of the PBB. As their name suggests, they take part in the

[14]https://gdpr-info.eu/recitals/no-26/

blockchain consensus, thereby producing new blocks and validating any new transactions. Each consensus node is connected to a number of other nodes in the network with which they exchange transactions and blocks (indicated by the scattered arrows in Figure 4.5). Each consensus node stores a copy of the whole blockchain and exposes web endpoints for clients to post and query data.

For liveness and security, Tendermint's consensus mechanism requires at least two thirds of all consensus nodes to run honestly. To accomplish such a situation, multiple independent parties interested in a smooth election and a valid result need to run consensus nodes. The set of consensus nodes should be fixed before the election begins, although technically, it is possible to remove existing ones and add new ones while the blockchain is running. The identities of the consensus parities must be publicly known, to be able to keep them accountable.

The second type of node is a full node. The only difference to the consensus nodes is that a full node does not take part in the consensus mechanism. Otherwise, they run the same server software. The difference is solely a configurational one. Full nodes get their name from the fact that they also store the whole blockchain history. They take part in the peer-to-peer protocol and can receive and answer requests from clients just like the consensus nodes. Anyone can run a full node and thereby directly observe the process of an election.

Administration Client

The administration client is meant to be used by the voting authorities. Potentially, the administration client can be used by the authorities to:

- Generate the election configuration and post it on the PBB.
- Authenticate voters that have posted their public credentials in the registration phase.
- Add authenticated, eligible voters to the list on the PBB.
- Compute polynomial coefficients and post it to the PBB.
- Define the election generator and post it to the PBB.

The prototype shifts most of these tasks to the PBB, as we will see in Chapter 5. Therefore, the administration client is mostly used to configure the consensus nodes and to set up the basic PBB network at the beginning of an election.

The administration client communicates with the PBB nodes over an HTTP connection, which only needs to provide integrity. It does not have to be private since the information posted by the voting authorities is public and known to originate from them.

Voter Client

The voter client application encompasses all actions a voter can take. That includes:

- Generate private and public credentials and post the public credential to the PBB.
- Check the list of eligible voters for the inclusion of own public credential.
- Generate and send all necessary parts of a ballot (election credential, commitments, ZKPs, and vote) to the PBB.

The voter client communicates with the PBB nodes over an HTTP connection. In the vote casting phase, this connection needs to be anonymous and is, therefore, run through an anonymity network like Tor (see Section 4.3.4). If the voter specifically queries her ballot transaction (for individual verifiability), the connection must go through Tor as well.

As discussed in Section 4.3.5, PBB nodes can try to respond to voter's requests with fake data. The affected voter will not see the correct picture of the PBB's contents, and depending on the information given will create an invalid ballot. For this purpose, the voter client should request a confirmation of the posted ballot via a separate PBB node. If the first node was malicious and the second node is honest, the voter client will recognize the oddity and can recreate the ballot. The other option is to use a separate device to check the posted ballot, as discussed in Section 4.3.6. This is also the practical solution to the untrusted voter platform. Malicious code on a voter's device can modify the voter's vote before posting it to the PBB. For this to be noticed by the voter, she needs to check the PBB using a separate device.

Verifier Client

The verifier client is an application allowing anyone to verify the correctness of the credential polynomial and final voting result. Verification, in this case, means re-calculating the polynomial and the election result and comparing it to the ones published on the PBB. The verifier client can:

- Calculate the credential polynomial from the list of eligible voters and compare it to the published polynomial.
- Verify all ballots, calculate the election result and compare it to the published result.

The prototype does not provide a separate verifier client but provides its functionality via the voter client.

External Systems

Figure 4.5 shows two systems outside of the voting system scope. For one, this is the IdM system that provides the voting system with information about the eligibility of voters (see Section 4.3.3). The second one is the CRS setup system that is an abstraction of the mechanism needed to generate a CRS as a basis for the group generators (see Section 4.3.7). Both systems are outside of the scope of this work.

Chapter 5

Implementation

This section documents the prototypical implementation of the UP protocol done in this work. All relevant source code is included in one Go module called `up-voting-system` available at Github[1]. The packages of the module and their dependency hierarchy are shown in Figure 5.1. Each of the three `cli` packages contains an entry point to a command line interface (CLI) application. The `cli/pbbd` package contains the entry point for the PBB application, *i.e.,* compiling it generates an executable that can be deployed on a prospective PBB node. The other two packages `cli/vcli` and `cli/acli` compile to a voter and a voting administration CLI application, respectively. All three `cli` packages depend on the `pbb` package that implements the PBB functionality and CLI commands that can be used to interact with the PBB. Finally, the `crypto` package implements the cryptographic primitives required by the UP protocol.

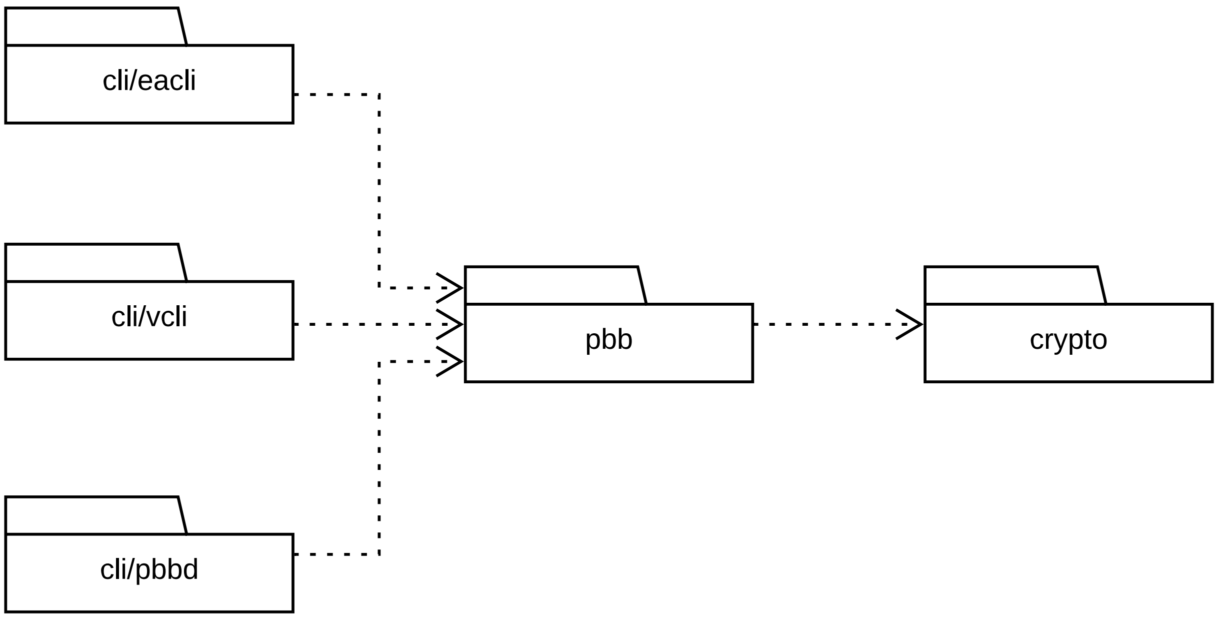


Figure 5.1: Source code package structure of the voting system prototype.

[1]`https://github.com/csmuller/up-voting-system`

As the PBB is central to the UP protocol, so is the `pbb` package central to the prototype. It implements a Cosmos-SDK module, meaning that it is structured similarly to how Cosmos-SDK implements other modules (*e.g.,* the `staking` module[2]). A Cosmos-SKD module defines a set of functionalities that can be added to a blockchain implementation. Modules can depend on other modules. A module can implement its own transaction types (*e.g.,* a transaction for registering a voter) and handlers that process these transactions. If the blockchain receives a transaction, Cosmos-SDK automatically hands it over to the correct module that knows how to deal with it. For each transaction type CLI commands or REST endpoints are implemented that allow sending and querying information related to that transaction. Each module has access to its own region of a data storage that backs the application. Modifications triggered by incoming transactions update the storage of the module. At the same time, the transaction recorded in the blockchain, making it possible to replay all changes in case the data storage is erased or a new blockchain node synchronizes with an already running network. Because the `pbb` package is a package in the language of Go but also a module in the language of Cosmos-SDK, it will be referred to as a package or module interchangeably.

Note that the prototype lacks a separate application for the verifier client mentioned in the system architecture in Section 4.4. This is because the verifier functionality is included in the two other client applications `vcli` and `acli`.

5.1 Cryptography

5.1.1 Implementation of Zero-Knowledge Proofs

The ZKPs are the centerpiece of the UP protocol. Though, their correct translation from formal notation into code is a challenge. As described in [Loc16], an implementation of the proof systems exists in the UniCrypt library[3]. The library is written in Java and can, therefore, not directly be reused in our Go-based prototype. A decision on how to implement the cryptography in the prototype had to be made. Two options were considered, (1) find a way to make the UniCrypt library available to Go programs or (2) re-implement the proof systems in Go.

The two most plausible ways of achieving option (1) are discussed in the following. *(i)* Implement a separate server that integrates the UniCrypt code, offers an RPC interface, and is called by the voting system's application via that interface. The server is packaged with whatever application relies on the proofs (*e.g.,* the PBB) and runs on the same machine. *(ii)* Use JNI[4] and Cgo[5] in combination as a double indirection to make UniCrypt callable from within Go code. Golang's Cgo allows mixing C code invocations with Go code while Java's JNI allows C code to call Java code (and vice versa). A Go library

[2]`https://docs.cosmos.network/master/modules/staking/`

[3]`https://github.com/bfh-evg/unicrypt`

[4]`https://docs.oracle.com/javase/8/docs/technotes/guides/jni/`

[5]`https://golang.org/cmd/cgo/`

called JNIGI[6] exists that removes the necessity of writing the intermediary C code and promises direct calls of Java methods from within Go code. Both *(i)* and *(ii)* introduce indirections with additional complexity and uncertainty for the development process and the applications. Moreover, the maturity and precise capabilities of the JNIGI library in *(ii)* are not known.

For option (2), internet research was conducted for Go libraries that are of use to the implementation of the proof system. No suitable resources were found for implementation of the set membership proof π_1 and the proof of knowledge of representation of committed value π_2. Though, an implementation for a proof of knowledge of discrete logarithm similar to π_3 was found in an open-source library called emmy[7] The library also offers an implementation of basic mathematical concepts like groups and commitment schemes.

The final decision was made in favor of option (2). It makes our implementation independent from UniCrypt. UniCrypt does a great job of mapping cryptography-relevant math into a software library. However, its many layers of abstraction make it opaque and hard to understand, *e.g.,* which group is being operated on just by reading the code. All proofs were re-implemented in Go, and the corresponding code in UniCrypt was used as the template. Abstractions are used as little as possible. The implementation of required mathematical concepts, *i.e.,* groups, generators, polynomials, and commitment schemes, are inspired by the emmy library but also re-implemented. The goal was to allow the reader to easily transition between the theoretical definition of the protocol and its implementation.

The fundamental mathematical operations like multiplication and exponentiation are not optimized. All math implemented in the prototype makes use of the integer type `Int` of Go's `math/big` package, and, therefore, of its operations.

5.1.2 Security Level

The verifiability properties of the UP protocol depend on computational intractability, at least as long as an election is running. Thus, the protocol parameters have to be set up such that a present-day attacker is computationally unable to break the protocol. If weak parameters are chosen an attacker can for example find α' and β' such that for some public credential of an eligible voter $u = g_1^{\alpha'} g_2^{\beta'}$. With these credentials, the attacker can post a ballot, *i.e.,* produce valid ZKPs, like any other eligible voter. The modulus and order of the groups $\mathbb{G}_p$ and $\mathbb{G}_q$ have to be chosen large enough to thwart such attacks. Additionally, the security parameter κ used in proof π_2 needs to be large enough to guarantee the soundness of the proof. I.e., the prover must not be able to make a verifier accept a wrong statement.

For orientation in how to choose the parameters, the NIST recommendations were consulted [GB20]. Table 5.1 shows different choices for the security parameter κ and the groups' order and modulus. Numbers in the order and modulus columns are denoted in

[6]`https://github.com/timob/jnigi`

[7]https://github.com/xlab-si/emmy

bit lengths. E.g., for a security level of 80, the group modulus should be a prime of 1024 bit length.

The argumentation for the group order size is as follows. The order of the group $\mathbb{G}_q$ determines the security of the private credentials α, β. I.e., the size of the order is the size of the keyspace for the credentials. Therefore we derive the order's bit size from recommendations on key size for DL-based cryptosystems [GB20].

Cryptoperiod	Security level	Group order	Group modulus	κ
Legacy-use	80	160	1024	80
2019 - 2030	112	224	2048	112
2019 - 2030 & beyond	128	256	3072	128

Table 5.1: Protocol parameter recommendations for different security levels.

A security level of 80 is not considered secure anymore by the NIST [Bar16]. However, because the UP protocol does only require computational intractability at the time of the election, a security level of 80 is probably still enough at the time of writing. Otherwise, NIST recommends using a security level of at least 112 starting from 2019.

The prototype has a few sets of ready-made protocol parameters hardcoded. The one that is used by default chooses the order q and the modulus p of $\mathbb{G}_q$ to be 160 and 1024 bit, respectively. The order p (equal to the modulus of $\mathbb{G}_q$) and the modulus o of $\mathbb{G}_p$ are 1024 and 1034 bit, respectively. The security parameter κ is 80.

5.2 Account Setup and Identity Management

The UP protocol requires the voters to register with a publicly known identity or at least one known by the voting authorities. This implies the existence of an external system that manages these identities. The prototype does not include such an IdM system and, therefore, does not allow to filter voter registrations by identity. Thus, anyone can register with any identity. Identity in this context means a key pair used to sign the registration transaction. A voter can create any number of new key pairs and registration transactions. It is part of future work to integrate an actual IdM system and add the corresponding identity checks in the PBB implementation.

While the UP protocol only requires an identity for the registration transaction, Tendermint technically requires identifying signatures on any transaction, but adding the voter's signature on the ballot transaction clearly breaks BP. Therefore, the prototype proposes that all voters use a shared and publicly available voter key pair to sign these transactions. This fulfills Tendermint's need for a signature but, at the same time, keeps the voter's identity hidden. In an operational voting system such a shared key pair could be hardcoded in the voter client.

Finally, we need to deal with the public and private credentials (u, α, and β). The voter client simply stores these credentials unencrypted on the voter device's file system and asks for their location when constructing a ballot. This is not a suitable solution for an operational voting system. It is part of future work to determine a suitable mechanism that handles the election-related credentials without burdening the voter, but also without adding new trust assumptions, *e.g.,* by introducing a central authority managing all credentials.

5.3 Voting Protocol

This section discussed more details of the implementation by running through the process of an election. The explanation is split into the protocol phases outlined in Section 4.1.1 with the addition of a configuration phase in the beginning. The prototype does not fully adhere to the theoretically defined phases but rather interleaves some of the phases. The sequence diagrams shown along with the explanations deviate from the ones shown in Section 4.1.1.

5.3.1 Configuration Phase

Before an election can start, the parameters required by the UP protocol have to be setup on the PBB. The protocol parameters include

- The modulus o and order p of group $\mathbb{G}_p$.
- The modulus p and order q of group $\mathbb{G}_q$.
- The generators $g_1, g_2 \in \mathbb{G}_p$ and $h_1, h_2, h_3 \in \mathbb{G}_q$.
- The election generator $\hat{h} \in \mathbb{G}_q$.
- The security parameter κ used in proof π_2.

In the original protocol, no process for setting up these parameters is described (except the election generator). Intuitively one would choose the voting authorities to coordinate this phase. However, as mentioned in 4.3.7, the choice of the group generators needs to happen in a trustless manner, but the prototype does not yet provide this such a setup. Whoever sets up the PBB network needs to choose the parameters and enters them in the source code of the PBB. The implementation adds them to the genesis block (see Figure 5.2). As its name suggests, the genesis block is the first block of the PBB blockchain, and it is created before the PBB runs for the first time. Every Cosmos-SDK module can specify a custom genesis state that is included in the genesis block. The prototype's pbb module uses the genesis block to set the protocol parameters. It defines its own genesis state in `pbb/internal/types/genesis.go`. The actual protocol parameter definition is found in `pbb/internal/types/params.go`.

Three sets of example group orders and modulus o, p and q with different bit lengths are listed in the `crypto` package in `crypto/groups.go`. One of these triples is used for the default parameters. The value for the security parameter κ is defined there too. The group generators g_1, g_2 and h_1, h_2, h_3 as well as the election generator $\hat{h}$ are chosen randomly from the respective group.

The protocol parameters can be queried but not changed once the blockchain is running. Although, the `governance`[8] module could be used to propose changes to them. While changing the protocol parameters in a running election should not be possible, this functionality could be part of a distributed setup of the CRS and thereby the group generators (see 4.3.7).

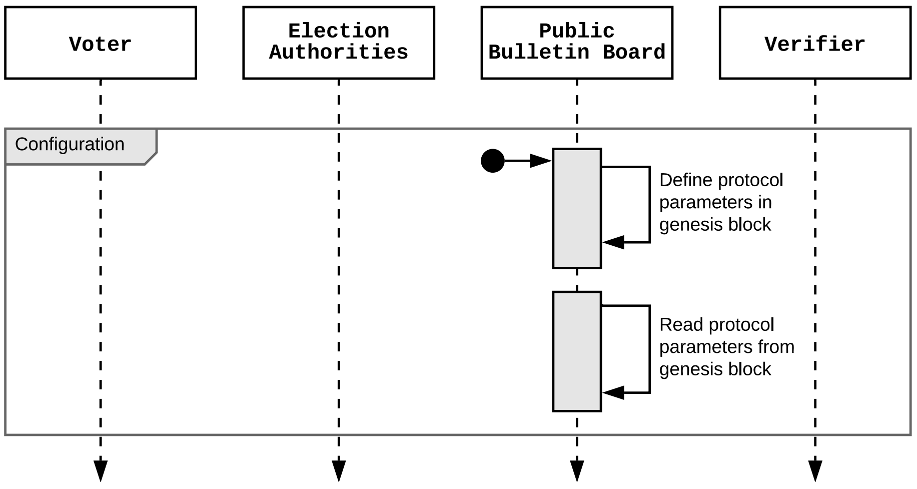

Voter
Election Authorities
Public Bulletin Board
Verifier
Configuration
Define protocol parameters in genesis block
Read protocol parameters from genesis block

Figure 5.2: The sequence diagram of the configuration phase as implemented by the voting system prototype.

5.3.2 Registration Phase

The registration phase requires each voter to create new credentials and post them to the PBB. At this point, all the information needed for that is available in the genesis block. A voter uses the CLI client compiled from the `cli/vcli` package to query the PBB for the parameters set in the genesis block. File `pbb/client/cli/query.go` contains all the CLI commands for queries that the PBB responds supports. With a local copy of the protocol parameters the client can randomly generate the private credentials α and β and from them the public credential $u = g_1^{\alpha} g_2^{\beta}$. The voter client stores the credentials locally and sends a transaction containing them to the PBB. With an IdM system in place, the voter would sign the transaction with her public identity (*i.e.,* the private key of her key pair), but this is not part of the prototype. Instead, the voter can use any key to sign the transaction. Cosmos-SDK includes CLI commands in the voter and administration client to create such key pers. The CLI command for creating the registration transaction is implemented in `pbb/client/cli/tx.go`. That file contains all the commands producing transactions supported by the PBB.

[8]https://docs.cosmos.network/master/modules/gov/

The registration phase, as described above, is congruent with how the protocol describes it. However, the prototype mixes in parts of the next phase, the preparation phase. The posted credentials are already processed in the registration phase directly on the PBB, as seen in Figure 5.3. The next section provides more information on this.

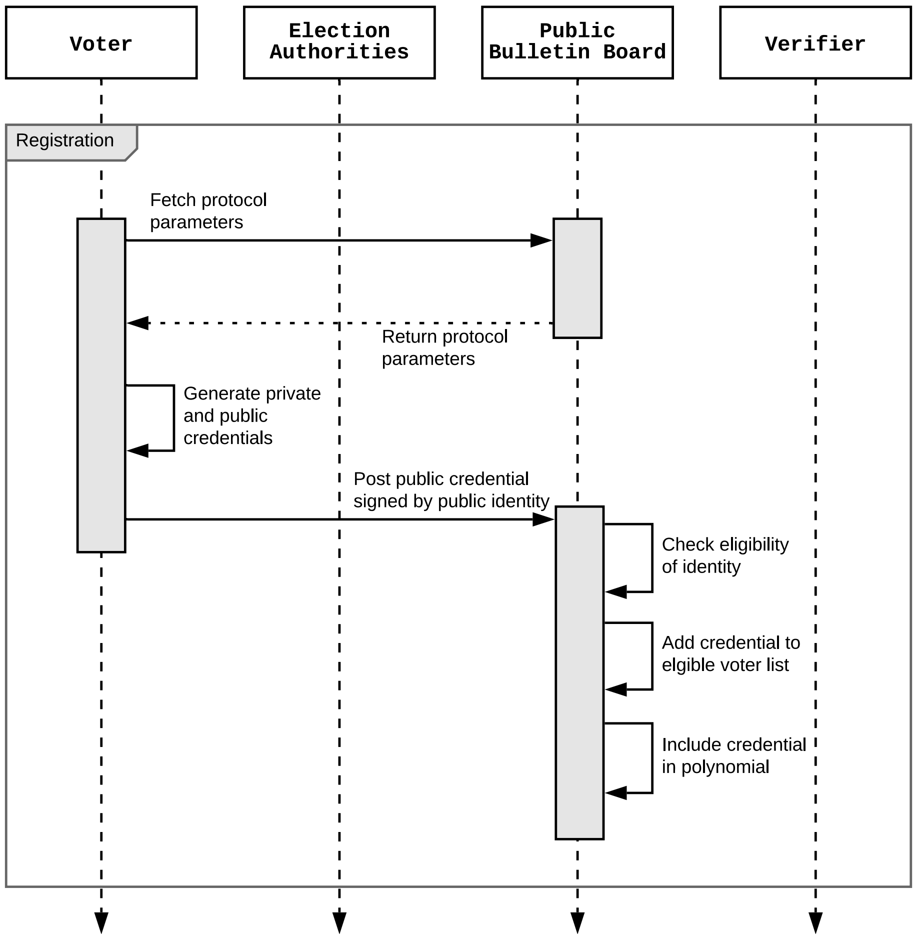

Voter
Election Authorities
Public Bulletin Board
Verifier
Registration
Fetch protocol parameters
Return protocol parameters
Generate private and public credentials
Post public credential signed by public identity
Check eligibility of identity
Add credential to elgible voter list
Include credential in polynomial

Figure 5.3: The sequence diagram of the registration phase as implemented by the voting system prototype.

5.3.3 Preparation Phase

The election preparation includes creating the list of eligible voters U, calculating the credential polynomial $P(X)$ with all public credentials and generating the election generator $\hat{h}$. The prototype spreads these activities over the previous two phases. The election generator is established in the genesis block with the other protocol parameters, and the voter list, as well as the credential polynomial, are continuously updated as new voters register (as seen in Figure 5.3).

When a voter sends a registration transaction to a PBB node, Tendermint routes the transaction to the `pbb` module, where it is handled by the logic in `pbb/handler.go`. This is where the procedures for all transactions are defined. In case of the registration the voter sends a transaction containing a `MsgPutVoterCredential` message. The public credential contained in the message is stored, and the credential polynomial is updated immediately. These and all other storage-related activities of a PBB node are implemented in `pbb/internal/keeper/keeper.go`. Updating the polynomial $P(X)$ with a public credential u happens by retrieving the polynomial from the storage, multiplying it with $(X - u)$, and writing the result back to the storage. All polynomial related functionality is implemented in `crypto/polynomial.go`.

After the preparation phase is over and all public credentials have been recorded and included in the credential polynomial, the polynomial should be checked for correctness. Anyone can retrieve the polynomial and the list of registered voters, re-compute the polynomial from the list and compare the result to the retrieved polynomial (see Figure 5.4). The CLI command for fetching the polynomial is implemented in `pbb/client/cli/query.go`.

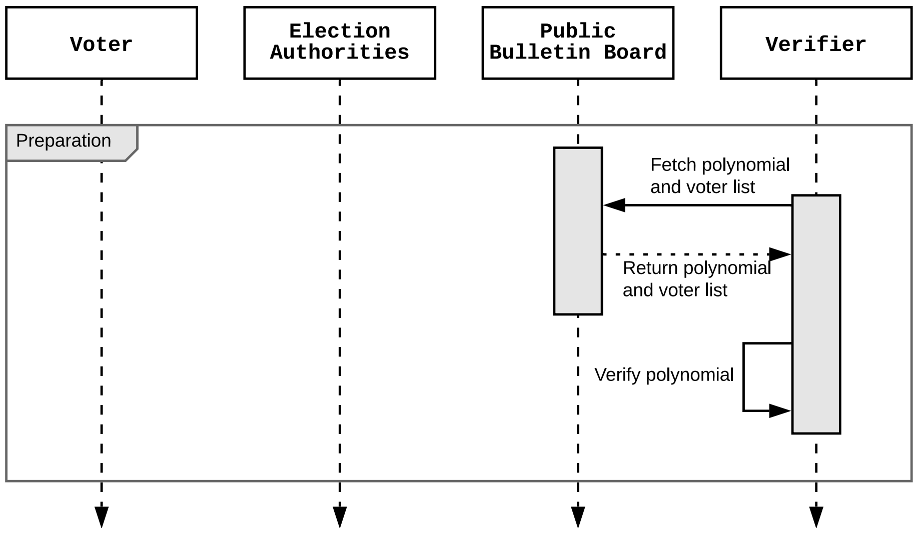


Figure 5.4: The sequence diagram of the preparation phase as implemented by the voting system prototype.

5.3.4 Vote Casting Phase

Before the voter client can generate a ballot, it needs to fetch the credential polynomial. The protocol parameters should already be locally available to the client from the registration phase. Having retrieved the polynomial, the voter should verify that its public credential is included by checking $P(u) = 0$.

The file `pbb/client/cli/tx.go` implements the whole process of generating a ballot and wrapping it into a transaction in one CLI command. The following steps are performed sequentially:

1. Compute election credential $\hat{u} = \hat{h}^{\beta}$.
2. Create commitment $c = com_p(r, u)$ to public credential u with random value r.
3. Create commitment $d = com_q(s, \alpha, \beta)$ to private credentials α and β with random value s.
4. Specify a string v containing the vote.
5. Generate the set membership proof π_1 with $P(X)$, u, r, c and v as inputs.
6. Generate the proof of known representation of a committed value π_2 with u, r, c, α, β, s, d and v as inputs.
7. Generate the preimage equality proof π_3 with $\hat{u}$, α, β, s, d and v as inputs.

The vote v is a plain text string. Its format depends on the election. Checks for adherence to a specific format could be implemented on the PBB or the voter client. All functionality for computing the commitments and the proofs resides in the `crypto` package.

The voter client sends the ballot to the PBB in a transaction containing a `MsgPutBallot` message. For this, the connection to the PBB should happen over an anonymous channel, *e.g.,* the client has to connect to the Tor network before sending the transaction. Furthermore, the client must not sign the transaction with the private key used in the registration phase. However, because Tendermint requires transactions to be signed, a generic key distributed with the voter client can be used.

On receiving a ballot transaction, the PBB node first checks if a ballot already exists with the same election credential $\hat{u}$. If so, the ballot is dropped, and no further verifications are made. As discussed in Section 4.3.2, voters are not allowed to post multiple ballots. This prevents attacks in which a lot of replayed ballots are posted from being effective. After this first check the proofs pi_1, pi_2 and pi_3 are verified in this order. If any of them are invalid, the ballot is dropped. If all verifications are successful, the ballot is stored.

Figure 5.5 depicts the vote casting phase.

5.3.5 Tallying Phase

When the election is over, the voter and administration clients can be used to query all ballots from the PBB and do the vote-tallying from the plain text votes. Technically it is not necessary to verify the ZKPs because the PBB already verified the ballots in the vote casting phase. Nevertheless, the clients redo the verification and keep only the votes of valid ballots for vote counting.

Figure 5.6 depicts the tallying phase.

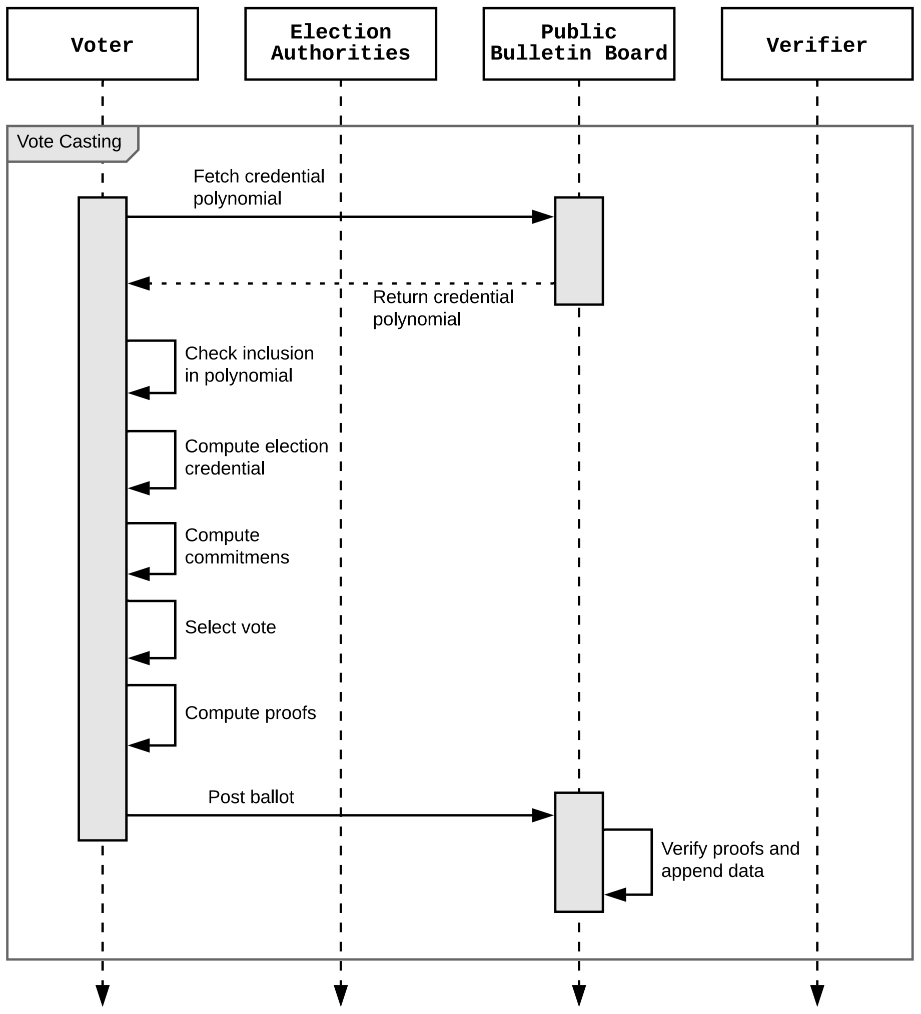

Voter
Election Authorities
Public Bulletin Board
Verifier
Vote Casting
Fetch credential polynomial
Return credential polynomial
Check inclusion in polynomial
Compute election credential
Compute commitmens
Select vote
Compute proofs
Post ballot
Verify proofs and append data

Figure 5.5: The sequence diagram of the vote casting phase as implemented in the voting system prototype.

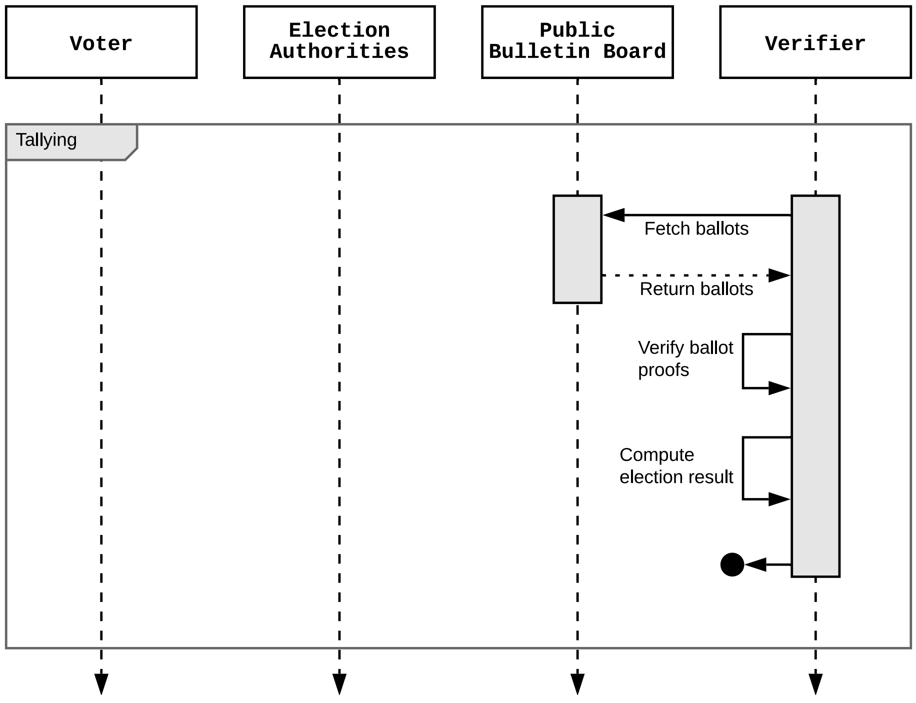


Figure 5.6: The sequence diagram of the tallying phase as implemented in the voting system prototype.

Chapter 6

Evaluation

This chapter describes the evaluation of the implemented voting system prototype. First, we analyze if the prototype effectively materializes the theoretical properties of the UP protocol. Second, a practical evaluation of the system is conducted, and its results documented. Third, by the insights gained from the evaluation, we shortly discuss possible use cases for the prototype.

6.1 Voting System Properties

The UP protocol promises individual, universal, and eligibility verifiability, as well as unconditional BP [Loc16]. This section evaluates if the prototype fulfills all underlying assumptions that are needed for these properties to hold. First, the assumptions are recalled for each property of the protocol. Second, the parts of the prototype implementing these assumptions are discussed.

6.1.1 Properties and Assumptions of the Protocol

Individual verifiability is based on the fact that a voter can look up her ballot on the PBB, thereby checking if it has been correctly included. The protocol assumes the existence of an append-only bulletin board that guarantees appended ballots cannot be removed or modified. The protocol also assumes that the voter can check the PBB from a trusted platform, meaning that no malware can compromise the voter's view on the PBB.

The UP protocolclaims to provide universal verifiability because anyone can take part in the final tallying process. The assumption that all ballots used in the vote-tallying are valid is grounded on the following aspects.

- The ballots are posted on an append-only PBB that guarantees votes cannot be modified or deleted.

- The ZKPs included in the ballots prove that the originator of a ballot is indeed eligible to cast a vote. The ZKPs are computationally sound, meaning that no computationally bounded adversary can create a valid ballot without owning eligible voter credentials.
- No computationally bounded attacker can compute the private credentials of another voter (DL assumption) and is thereby unable to post votes in the name of other voters.
- The generators of the groups $\mathbb{G}_p$ and $\mathbb{G}_q$ are independent, respectively, preventing voters from casting multiple valid ballots under the same public credential (see Section 4.3.7).

By the definition found in [Loc16], eligibility verifiability is included in universal verifiability, but in our work, we defined it separately. The UP protocol establishes eligibility verifiability first by setting up a list of eligible voters in the preparation phase and later by having the voters include ZKPs in the ballots. The soundness of the ZKPs prove that the originator of a ballot is indeed eligible to cast a vote. What remains to be clarified is the nature of the voter list. The protocol does not specify where the identities for this list come from, only that the voting authorities produce it. That aside, the voter list is publicly available on the PBB, meaning that anyone can check for inclusion and complain if something is wrong.

The protocol has unconditional BP because firstly, not even a computationally unbounded adversary can link ballots to voters, and secondly, the privacy does not depend on trust in a third party (*e.g.,* on voting authorities with a shared secret). The first statement is based on the fact that ballots are only published with a perfectly-hiding commitment of the corresponding public credential and ZKP that are perfectly zero-knowledge [LH15]. Therefore, no information about the voter is given away when posting a ballot. Additionally, an anonymous channel is assumed that obscures the voter's identity on the network. The second statement is true because no third party is needed to protect the ballot. It is anonymous without the need for encryption with a public key of a third party.

6.1.2 Realisation of the Properties and Assumptions

The PBB implementation required by the protocol is at the core of the prototype. It is based on blockchain and a BFT consensus mechanism provided by Tendermint and provides the append-only property. The blockchain's data structure makes sure that no one can make modifications to a ballot without requiring changes to all subsequent blocks. Because the UP protocol theoretically requires no trust in a central authority nor multiple authorities for its properties, the PBB should also only introduce minimal trust assumptions in practice. Blockchain is an ideal solution because its decentralized network guarantees that no single entity has sovereignty over the PBB data. Tendermint's BFT consensus mechanism minimizes the required trust in PBB operators. Only if more than one third of all validators are malicious, the PBB can be brought to a halt, and only if two thirds or more are malicious, invalid data can be appended [BKM19]. The distributed structure of the prototype's PBB also provides robustness. Single node failures do not lead to failure of the whole PBB. Similarly, single nodes acting maliciously by responding to clients with fake data and not correctly processing incoming data can be spotted and bypassed due to the availability of multiple nodes (see Section 4.3.5). We conclude that the prototype's PBB fulfills the protocol's assumptions.

The PBB provides the first foundation for individual verifiability. However, the prototype cannot provide the second assumption of a trusted voter platform. This assumption is impractical with today's electronic consumer devices, and implementations of the UP protocol have to find other ways to deal with it. The prototype does not implement any extra measures to mitigate the problem of a compromised voter platform. However, voters can check the PBB with a separate device if they want to make sure that their transactions were not modified by malware (see Section 4.3.6). With that, individual verifiability is provided in practice.

Universal verifiability relies on the ZKPs for proving eligibility of the voter behind a ballot. The prototype implements all three ZKPs, as defined in the protocol. Therefore, the prototype's proofs can be expected to be computationally sound, making it infeasible to create fake proofs. Furthermore, the PBB implementation guarantees that no ballots on the PBB have been removed or modified. Thus, a verifier fetching all ballots from the PBB to do the vote-tallying can be sure of their integrity, that they originate from eligible voters, there is only one valid ballot per voter, and no posted ballots are missing. The only thing that is missing in the prototype, but assumed by the protocol, is a trustless setup of the group generators. Voting authorities setting up the prototype can choose generators for which they know the relative discrete logarithms. They can then register an eligible voter account and post multiple ballots under that account because the commitment to their public credential is not computationally binding for them. We conclude that the prototype achieves universal verifiability but with weakened trust assumptions. I.e., trust is needed in whoever instantiates the group generators.

Concerning eligibility verifiability, the prototype implements the necessary cryptography but does not yet deliver a ready-made solution for the voter identities. The prototype assumes that every registration transaction is from an eligible voter. Voters can use any key pair to sign their registration transactions. Thus, anyone can register for an election, also multiple times with different credentials. As mentioned in Section 4.3.3, a separate IdM system should be used as an oracle to the PBB, allowing the PBB nodes to check if the signature on a registration transaction belongs to a voter listed in the IdM system.

Finally, the prototype implements unconditional privacy. The implementation hides the voter's public credential in a perfectly-hiding Pedersen commitment and produces ZKPs that are perfect zero-knowledge. A ballot created with the prototype does, therefore, not reveal any information about the voter. Though, the anonymous channel, assumed by the protocol, is not part of the prototype. Without using a separate connection through Tor or a similar anonymity network, BP can be compromised. Assuming that ballots are at least send via encrypted channels to the PBB nodes, the most accessible location to link a network address to a ballot is on the PBB nodes themselves.

In conclusion, the prototype realizes the verifiability and privacy properties promised by the protocol but partially depends on external systems. Without a trustless setup for the group generators, the universal verifiability requires trust in the voting authorities for setting up independent generators. Without an anonymous channel, BP is compromisable. Without an identity management system, anyone can register for an election and do so multiple times.

The voting system properties Fairness, Receipt-Freeness, and Coercion-Resistance have been excluded from all discussions so far, and our prototype does not support these properties. In [Loc16], extensions to the UP protocol are proposed that add these properties without weakening the trust assumptions for unconditional privacy. The extensions lead to a more complex system overall.

6.2 Performance and Scalability

The UP protocol's desirable privacy and verifiability properties come at the cost of much computation. Generating and verifying the ZKPs is the main bottleneck of the protocol. Before going into the practical evaluation, the time and space complexity of the proofs are discussed.

The generation of the set membership proof π_1 requires $O(log(N))$ exponentiations in $\mathbb{G}_p$ and $O(N\ log(N))$ multiplications in $\mathbb{Z}_p$ and the verification $O(log(N))$ exponentiations and $O(N)$ multiplications in $\mathbb{G}_p$. The proof transcript contains $O(log(N)$ elements from $\mathbb{G}_p$ and $\mathbb{Z}_p$ respectively [LH15]. N is the size of the electorate, and $\mathbb{Z}_p$ is the set of integers modulo p. Note, that $\mathbb{Z}_p$ and $\mathbb{G}_p$ do not share the same modulus. The differentiation in which group an operation happens is important because the group's modulus decides on the possible size of elements in the group and therefore influences computational efforts. The generation and verification of the proof of known representation of committed value π_2 require $O(\kappa)$ multiplications in $\mathbb{G}_{\text{I}}$ and $\mathbb{G}_{\text{II}}$ respectively [ASM10]. The transcript size depends on κ linearly. Finally, The preimage equality proof π_3 has a neglectable influence on the time and space complexity.

The computational effort for generation and verification, as well as the size of the transcripts, depends on the groups' modulus and order. For example, if the bit length of the modulus o and order p of $\mathbb{G}_p$ is doubled, the proof transcripts of π_1 is also doubled because the elements in the transcript are elements of $\mathbb{G}_p$ and $\mathbb{Z}_p$. The positive effect is an enhancement of the security level. Because the choice of protocol parameters needs to provide security only during an election, the requirements of the UP protocol to the cryptoperiod of the parameters are low.

For the performance evaluation, the protocol parameters were chosen to provide a security level of 80, according to NIST (see Section 5.1.2). The order q and the modulus p of $\mathbb{G}_q$ are 160 and 1024 bit respectively. The order p (equal to the modulus of $\mathbb{G}_q$) and the modulus o of $\mathbb{G}_p$ are 1024 and 1034 bit respectively. The security parameter κ is 80. Tests were run on a Ubuntu 18.04.4 virtual machine with eight 2.4 GHz Intel Xeon E312xx CPUs, 16 GB RAM. Tendermint recommends at least 2GB RAM and a 2 GHz CPU with two cores. A small network of two consensus nodes responsible for the PBB was used, both running in their own Docker[1] container. Voter and voting authority clients were launched on the same machine for all transactions. The performance evaluation is divided by the transaction type that is under examination.

6.2.1 Ballot Transactions

The ballot transaction is the most prominent transaction of the UP protocol because it carries the vote. It is also its problem child because of its computationally intense ZKPs. The computations related to the ZKPs affect the voter client and the PBB. The client has to generate the proofs, and the PBB has to verify them. Although the UP protocol does not specify that the proof transcripts have to be verified by the PBB, this is the only way to avoid the PBB being stuffed with invalid ballots (see Section 4.3.1).

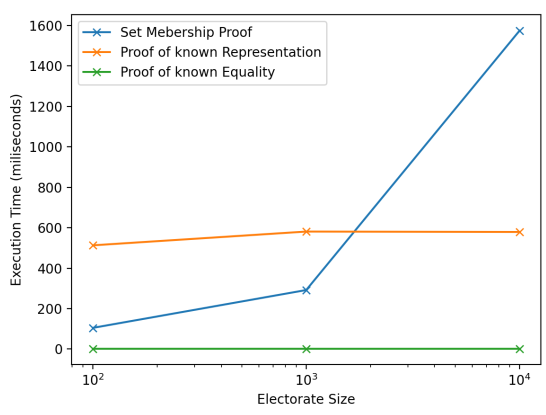
To evaluate the behavior of the proof generation and verification, three scenarios with different electorate sizes (100, 1'000, 10'000) were constructed. In each scenario, the required number of voter credentials were created and registered on the PBB. Then the credential polynomial was fetched from the PBB, and ballots were created for each voter.

Transaction Generation and Verification

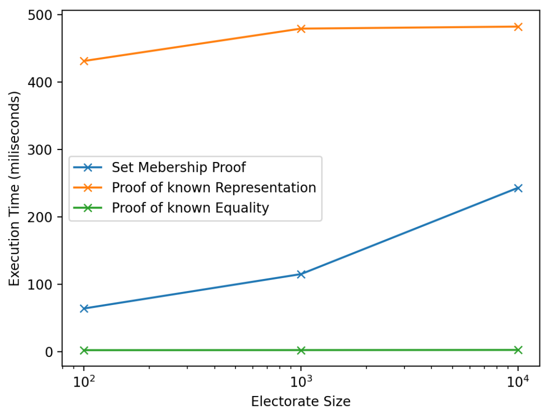
Figure 6.1a depicts the recorded runtimes for proof generation. The plot shows for each proof the average time the voter client took to produce a valid proof transcript. For clarity, we repeat the proof names and symbols used in the following text and the figures. π_1 is the set membership proof, π_2 is the proof of known representation (of committed value), and π_3 is the proof of known equality (of discrete logarithms). Effort spent on proof π_3 is neglectable as expected. Proof π_2 causes an almost constant amount of work that is unaffected by the electorate size. As mentioned above, it is only dependent on the security parameter κ. Increasing the security level of π_2 by increasing κ will make the time needed for proof generation grow linearly with κ. Proof π_1 is the only one that poses a scalability problem for proof generation. It seems to be rising linearly (note, the x-axis is in logarithmic scale), which is congruent with the linear component in $O(N\ log(N))$, multiplications in $\mathbb{Z}_p$.

Figure 6.1 shows the accumulated runtime for generating all three proofs, which are in line with the results achieved in [LH15]. There the poof generation took 1.4, 1.6, and 3.0 seconds for an electorate of 100, 1000, and 10'000, respectively. Although it is not entirely clear which group modulus and orders where used in their evaluation, we assume that they used the same as we did because they were the default setting in the UniCrypt code. Both UniCrypt and our prototype do not apply any specific optimizations to the proof generation. In [BG13] an optimized version of the set membership proof is evaluated, which achieves significantly lower runtimes, namely 24, 41, and 182 milliseconds for each electorate size, respectively. However, that evaluation is based on a smaller order p for $\mathbb{G}_p$ of 256 bit and a larger modulus o of 1536 bit, compared to 1024 and 1034 in our evaluation. For a direct comparison, tests with equal bit lengths have to be conducted.

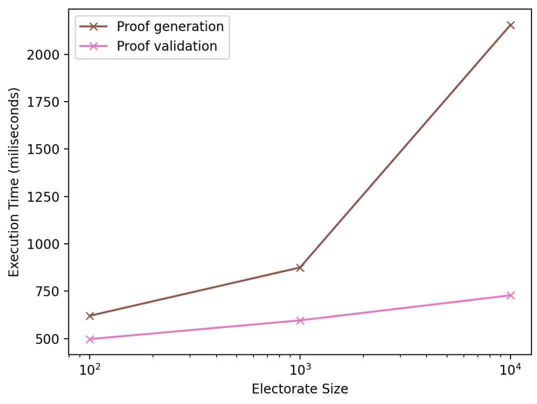
[1]`https://www.docker.com/`



(a) Average runtime of proof generation per proof.



(b) Average runtime of proof verification per proof.



(c) Total average runtime for proof generation and verification.

Figure 6.1: Average runtimes of proof generation and verification.

On the side of proof verification, the runtimes are also behaving as expected (see Figure 6.1b). The verification effort for proof π_3 is neglectable. For π_2, it is more or less constant, and for π_1, it increases with the size of the electorate. Because of the small number of data points, we are not able to tell if the increase is linear or logarithmic. We know that for a large enough N, the $O(N)$ multiplications will dominate over the $O(log(N))$ exponentiations, but we cannot tell if that already happens before $N = 10'000$. Figure 6.1 depicts the total runtime for proof validation. Again the results behave similarly to the ones presented in [LH15], where proof verification took 1.0, 1.1, and 1.3 seconds.

Although these runtimes might look promising, they are a major problem for the PBB and its consensus mechanism. The following are observations about the PBB's behavior when receiving a constant stream of ballot transactions.

Starting with only 100 registered voters, we loaded the PBB with about two ballot transactions per second (TPS). In the first few seconds, the nodes produced blocks of about 40 transactions in the usual block time of 5 seconds (by configuration). However, soon the block production stagnated and came to a complete halt. The logs show timeouts in the consensus mechanism. Only sometime after the 100 transactions have been sent out, new blocks were generated again. That was about 3 minutes after starting to send transactions and approximately 2 minutes after the last transaction was sent. Nevertheless, the PBB managed to add all transactions into blocks. With 1000 registered voters, the observations are similar, but anomalies start at a lower TPS. At one TPS, the PBB already shows increased block time but still manages to form blocks at regular intervals. With 1.5 TPS, timeouts in the consensus mechanism start to occur, and block time goes up to minutes. After the flooding stops, the consensus mechanism recovers, and all transactions are included in blocks.

An examination of the nodes' CPU usage shows an interesting pattern. After the point where the consensus does not progress anymore, there are times in which both nodes have moderate to low CPU loads, and the logs show a lot of data exchange between them. The data exchange by itself already takes more time than the usual 5 second block time. Before and after the exchange, the nodes take turns validating the proofs. In these periods, only one node at a time shows a high CPU load while the other seems to be waiting. These periods are much longer than the usual block time. This pattern continues for some time after the transaction flow stops, and then the PBB recovers and goes back to normal.

In conclusion, the consensus can be brought to a halt by flooding the nodes with ballots at a relatively low rate, depending on the electorate size. For 100 registered voters, that rate was at about 2 TPS and for 1000 at about 1.5 TPS. This issue makes the PBB an easy target for DoS attacks. If the flooding does not persist for long, the consensus continues correctly and succeeds in processing and including all transactions. However, if an attacker decides to flood the PBB with a much higher TPS and for a longer time, the transaction pools of the PBB nodes will fill up quickly. Because the consensus is not producing new blocks the transactions stay in the pool, and new transactions are rejected after some time. The cause of all of this is the runtime of proof verification, which makes the consensus nodes spend too much time on transaction validation, leading to timeouts in the consensus mechanism. At the same time, the transactions' size seems to impede the peer-to-peer communication between nodes, again leading to timeouts in the consensus.

Transaction Size

The improvement that stands out the most in the set membership proof proposed in [ASM10] is the logarithmic behavior of the proof transcript size when increasing the set size. Observation of the ballot transaction size in our prototype confirms this behavior. Figure 6.2 shows byte sizes for ballot transactions generated in the three different electorate setups. Mind that the x-axis is in logarithmic scale, which makes the logarithmic transaction size progression appear linear. The depicted sizes are taken from the transactions in their JSON format, meaning that their transfer size on the wire might be smaller, depending on how Tendermint deserializes the data. The transactions sizes were recorded before and after signing.

On the PBB nodes, the transactions seem to be taking less space than the JSON format. One of the databases that Tendermint creates on the PBB nodes is the `blockstore.db` which represents the primary block storage. It holds all the blocks, some metadata, and a block index. Its size was about 94 MB after casting 1000 valid ballots, which is less than the sum of these ballot transactions in their JSON format.

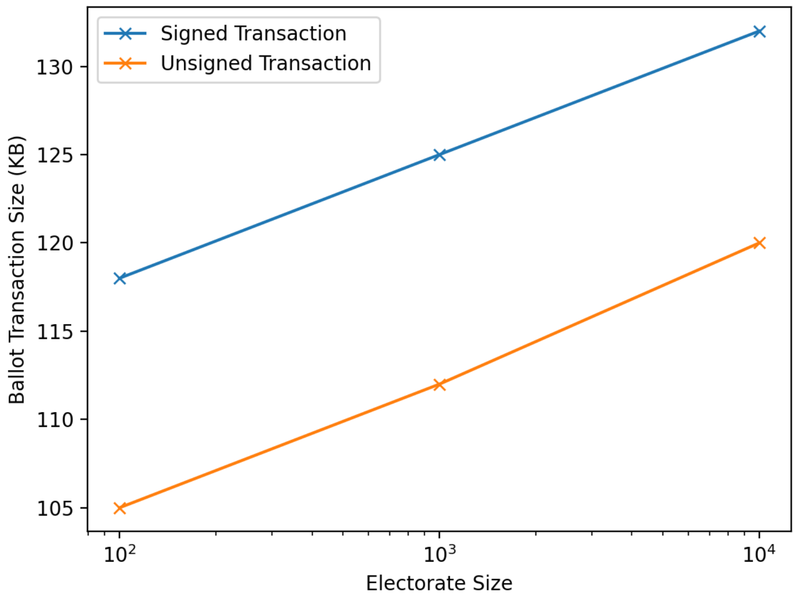


Figure 6.2: Size of ballot transactions dependent on electorate size before and after signing them.

Compared to the findings in [LH15], where 1000 voters produced 43.2 MB of ballots, our prototype produces about twice as much data. Scaling that up to 1'000'000 voters we can expect an accumulated size of about 100 GB, which is not a problem by today's standards.

6.2.2 Registration Transactions

Another computational intensive aspect of the protocol is the credential polynomial. The polynomial is calculated from all voters' public credentials. In the protocol, this calculation is performed by the voting authorities, either incrementally as ballots come in or after the registration phase is over. The prototype however, updates the polynomial on the PBB directly without the need for the authorities. By the properties of the blockchain, this ensures a valid polynomial. Theoretically, no recalculation and comparison are necessary by a verifier. The issue with this approach is the computational effort for the PBB nodes that increases as more and more voters are registered.

In the experiments we ran, the load of recalculating the polynomial with each new public credential became a performance bottleneck quickly, preventing the consensus mechanism from running smoothly. For example, after 1000 credentials were registered, the consensus mechanism started to time out when continuing to send new registration transactions at a rate of 7 TPS. I.e., with a polynomial of order 1000 and a TPS of 7, the insertion of new credentials into the polynomial became so intensive that the consensus mechanism came to a temporary halt. It recovered and continued normally as soon as the flooding with registration transactions stopped.

6.2.3 Mitigation Strategies

Compared to the issues observed with ballot transactions, the registration transactions are only a minor problem, and a feasible mitigation is available. First, effortless DoS attacks are not possible here, because incoming registration transactions are checked for their signature and eligibility before they are added to the polynomial. Flooded transactions from an attacker without access to eligible private keys will be dropped at the signature check. Second, the automatic update of the credential polynomial can be removed from the PBB's functions and shifted into an external process. I.e., the polynomial calculation is transferred to the voting authorities as initially foreseen by the protocol.

The ballot verification poses a major threat to the availability of the PBB. As discussed in Section 4.3.1, we cannot resort to a partial verification. Proofs π_1 and π_2 are both necessary to keep an adversary from posting invalid ballots, and leaving out π_3 does not make a performance difference. We should also not move the proof verification outside of the PBB because that allows anyone to spam the PBB's storage with any number of invalid ballots. Other ways have to be found to mitigate the voting system's weakness.

The prototypes proof implementation is not optimized, so more efficient implementations of the ZKPs will shorten proof validation and take some load off the PBB nodes. Though, this does not change the proofs' runtime complexities and therefore does not help with scalability. Real improvement of scalability can only be achieved by replacing π_1 and π_2 with more efficient proofs. However, to our knowledge, more efficient proofs are not available.

Besides touching the ZKPs, one can try to manipulate the blockchain configuration. The evaluation has shown that consensus nodes fill blocks up to the block size limit when flooded with transactions. This intuitively makes sense and is at about 200 transactions per block in our experiments, given a default block size limit of approximately 22 MB. The nodes transaction memory pools, set by default to a limit of 5000 transactions and 1074 MB, were never completely filled. Reducing these parameters might help the consensus to retain its liveness even in case of a high transaction rate. The idea is that with a smaller maximum block size, consensus nodes might be able to verify all proofs in the block without the consensus timing out. Evidently, reducing the transaction pool's capacity will lead to incoming transactions being dropped earlier, but might thereby protect the consensus from stagnating. Parameters of the consensus mechanism itself can also be configured (*e.g.,* timeouts) as documented on the Tendermint web site[2]. An alternative configuration of consensus timeouts might lead to a more stable PBB.

Adding supplementary infrastructure to the PBB is another option. That is, install sentry nodes that pose as filters in front of the actual consensus nodes. Such a setup is recommended by the Tendermint project[3]. Consensus nodes are hidden behind sentry nodes and not publicly reachable. Meaning, their identity is known, but their direct internet

[2]See `https://docs.tendermint.com/master/tendermint-core/running-in-production.html#configuration-parameters` and `https://docs.tendermint.com/master/tendermint-core/configuration.html#consensus-timeouts-explained`

[3]`https://docs.tendermint.com/master/tendermint-core/validators.html#setting-up-a-validator`

address is not. The task of the sentry nodes is to perform the proof validation and drop invalid ballots immediately. In the case of a DoS attack, the consensus nodes will not have to deal with the proof validation of invalid ballots, and block production can continue normally. The performance bottleneck shifts to the sentry nodes and does not affect the consensus mechanism anymore. By scaling up the sentry nodes in numbers and resources, some relief is given in a DoS attack. This approach does not fix the problems that arise when too many voters cast votes in rapid succession. All the valid transactions still make it to the consensus nodes and impede the consensus mechanism if received at a high enough rate. Finally, for the sentry node approach to work, every consensus node needs to follow the setup. If one consensus node starts including invalid ballot transactions in block proposals, these transactions also have to be handled by the other consensus nodes making the filtering of the sentry nodes useless.

6.3 Use Cases and Applicability

In consideration of the performance and scalability problems, the prototype is not ready to be used for nation-wide voting. One might argue that for such elections, where a trustable central authority exists, the protocol's strict trust assumptions are not necessary. Nevertheless, for situations where no established structure and central authority are available, the prototype is a great option. Starting with the instantiation of the protocol parameters, through to the construction of the PBB blockchain and the vote-tallying, every aspect of the prototype can and should be handled in a decentralized manner. A set of independent actors can set up and run an election without requiring trust in each other.

For example, an open-source community with the need for community polls can invite community members to apply as consensus nodes for the PBB. With a sufficient amount of independent participants, the PBB's safety and liveness are assured. The same participants can take part in the distributed setup of the protocol parameters. Although some prominent actor is required to initialize the PBB blockchain and set the agreed on parameters, all actions of that actor are publicly visible and malicious behavior is detectable. For establishing a list of eligible voters, such a system can use a separate blockchain-based IdM system, like SeraphID[4].

Even though the prototypes consensus mechanism struggles with the computational requirements of the UP protocol we think that blockchain is still the right technology choice. It satisfies the need for a distributed setup and is the natural successor of the work carried out on the topic of public bulletin boards (see Section 3.5). However, for further advances of this protocol and voting system, an alternative to Tendermint could be considered for the PBB. Since the most significant issues of the prototype arise in the consensus mechanism, a more in-depth analysis of this aspect is required and might reveal more fitting consensus mechanisms.

We propose Hyperledger Fabric as the next most promising candidate because it applies a transaction processing flow that is different from most other blockchains. The concept

[4]`https://www.seraphid.io/`

of collecting endorsements before a transaction is sent to the consensus mechanism takes the load away from the validators and places it on the endorsers. Fabric even allows the implementation of chaincodes/smart contracts in Go, which means that the prototype's ZKPs, the most difficult part of the implementation, can be easily ported. Moreover, even if a platform based on another language is chosen, the ZKPs are relatively easy to re-implement because their current implementation is without abstractions and because of Go simple to read.

Chapter 7

Conclusion and Future Work

In this thesis, a prototype of a REV system with unconditional privacy was designed, implemented, and evaluated. The underlying motivation was the rarity of practical implementations of voting protocols, especially of one with minimal trust in a central authority or a group of authorities. Different REV protocols were studied and assessed according to common voting system properties and the techniques used to achieve those properties. A promising protocol candidate was found in the work of Locher and Haenni [LH15; Loc16]. With that protocol and the decentralization capabilities of blockchain in mind, we set out to design a voting system that stays true to the theoretically achieved properties of the protocol as much as possible.

As one of the cornerstones of the chosen voting protocol, and for that matter of many others, the PBB required much attention in the design and implementation of our prototype. The technology for its realization was set from the beginning to be blockchain. Thus, the set of the PBB's particular requirements to a blockchain was elaborated, and several existing blockchain platforms were compared, leading to Tendermint and Cosmos-SDK as the final choice.

The second foundation of the protocol is cryptography and, in particular, ZKPs. After the decision was made to re-implement the required proofs, a sufficient understanding of the underlying mathematical basics was acquired and documented. The proofs were then implemented in Go, along with the rest of the prototype and successfully tested.

To evaluate the proper operation, performance, and scalability of the prototype, a small test network was set up, and experiments with different electorate sizes were performed. The insights from the evaluation lead to multiple suggestions on how to improve the system's performance and stability. Next to the practical evaluation, a theoretical examination of the prototype's accomplished properties compared to the protocol was conducted.

The most significant contribution of this work is the prototype implementation. It shows the practical feasibility of a blockchain-based, decentralized voting system with minimal trust assumptions. The usage of Go for its implementation and the refrain from opaque abstraction makes the source code relatively easy to understand and relate to the documentation given in this report. However, the evaluation of the prototype has shown that performance and scalability are an issue. The availability of the system can be threatened even with small transaction rates. Several mitigations for this problem are suggested in this work and will have to be considered in future work.

7.1 Future Work

The implementation developed in this thesis is intended to be a prototype and not a full-fledged voting system. Therefore, many aspects of the implementation can be improved and extended. Similarly, the conducted evaluation of the prototype can be continued, further exploring the behavior of the voting system under different circumstances. This section gives ideas for future work.

There are several features that can be added to the prototype for completeness.

Voting period. Provide the voting authorities with the possibility to set a start and end date for the voting period, making the PBB only accept ballots in that period and dropping them otherwise. This could be developed by simply adding the dates as parameters to the genesis block of the PBB, or by extending the implementation with a new transaction type.

Calculation of the credential polynomial. As shown in the evaluation, updating the credential polynomial directly on the PBB with every incoming registration transaction becomes a performance bottleneck for large electorates. Therefore, this calculation should be included in the voting administration client. Additionally, a new transaction type is then needed to upload the polynomial to the PBB.

Pagination for ballot queries. For large electorates, querying the list of ballots becomes problematic because of its size. Therefore a pagination feature should be added with which ballots can be requested in manageable subsets.

Querying multiple PBB nodes. Allow the voter client to fetch data from multiple PBB nodes, thereby circumventing the problem of being provided with wrong information from one malicious node.

Then there are bigger features that constitute whole new components and extend the prototype into a more complete and elaborate voting system.

Browser-based voter client. In a production-ready voting system, the voter client cannot be a simple CLI application. The most convenient and platform-independent realization of a GUI client is a web application. On the side of the PBB, this only requires adding a few JSON-RPC endpoints for which the backend functionality already exists. The client, on the other hand, will probably require re-implementation of the ZKPs and repeated performance evaluation thereof. Also, the handling of voter credentials needs to be reconsidered in a volatile setting, such as a web browser.

Anonymity network. The prototype requires an anonymous channel for its privacy properties. Instead of relying on an external anonymity network like Tor, a custom onion network between the blockchain nodes could be set up that tunnels ballots through several nodes, each peeling off a layer of encryption.

Optimization of ZKP implementation. Re-implement the ZKPs with the support of efficient math libraries. A starting point for this can be the implementation of the set membership proof in [BG13], which runs considerably faster than our implementation. Alternatively, the work in [HLG20] gives insight into which optimizations are possible.

Enhanced usability for individual verifiability. The prototype offers no convenient features for checking if a ballot was correctly recorded on the PBB. A separate voter client application should be implemented that allows a voter to verify her ballot on a second device, *e.g.,* by scanning a QR code on the device used for ballot casting.

Multi-party computation of a CRS. The setup of the protocol parameters requires that the generators of the groups $\mathbb{G}_p$ and $\mathbb{G}_q$ are independent, respectively (see Section 4.3.7). In order that this setup does not rely on trust in a single entity, a multi-party computation protocol could be applied that produces a CRS that, in return, is used to instantiate the group generators.

Identity Management System. The UP protocol relies on the existence of an IdM system that can provide information about voters and their eligibility to vote. Such a system is not part of the prototype and should be developed separately, desirably based on a decentralized system too.

Management of election credentials. Besides a voter identity (*i.e.,* public key pair), the UP protocol also introduces election specific credentials that are created by the voter before registering for an election. A practical voting system should not require the voter to handle these credentials by herself. Therefore, a separate system is needed that takes this burden from the voter and provides convenient and transparent credential management.

Further evaluation of the prototype can be conducted under the modulation of the following system aspects.

- Experiment with different security levels, *i.e.,* change the bit length of the group modulus and orders, as well as the security parameter κ, to evaluate the impact on the performance of the proof generation and verification.
- Setup a larger blockchain network with multiple validators and full nodes that are geographically distributed.
- Experiment with the configuration of the consensus mechanism. Tendermint's consensus mechanism can be configured in a few ways that might help make the PBB more stable in the face of transaction flooding.

Finally, the most involved suggestion implies the re-implementation of the entire prototype on a different blockchain platform. Because the prototype's main weakness lies in the performance of the consensus mechanism, a different blockchain with a different consensus mechanism than Tendermint might be needed. In the context of the performance issues, the requirements to the PBB blockchain stated in Section 4.2.4 might need reconsideration to open up to other blockchains. As mentioned in Section 6.3, Hyperledger Fabric constitutes an attractive alternative because of its different transaction processing flow.

Bibliography

[ABJO+10] Arne Ansper et al. *E-voting concept security: analysis and measures.* Tallinn, 2010.

[Adi08] Ben Adida. "Helios: Web-based open-audit voting". In: *Proceedings of the 17th USENIX Security Symposium* (2008), pp. 335–348.

[AKRS+13] Elli Androulaki et al. "Evaluating user privacy in Bitcoin". In: *Lecture Notes in Computer Science* 7859 LNCS (2013), pp. 34–51. ISSN: 03029743. DOI: 10.1007/978-3-642-39884-1_4.

[ASM10] Man Ho Au, Willy Susilo, and Yi Mu. "Proof-of-Knowledge of Representation of Committed Value and Its Applications". In: *Information Security and Privacy. ACISP 2010. Lecture Notes in Computer Science* 6168 (2010), pp. 352–369. DOI: 10.1007/978-3-642-14081-5_22.

[Bar16] Elaine Barker. *Recommendation for Key Management – Part 1: General.* 2016. DOI: 10.6028/NIST.SP.800-57pt3r1. URL: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf.

[BCPS+11] David Bernhard et al. "Adapting helios for provable ballot privacy". In: *Lecture Notes in Computer Science* 6879 LNCS (2011), pp. 335–354. ISSN: 03029743. DOI: 10.1007/978-3-642-23822-2_19.

[BDG13] Johannes Buchmann, Denise Demirel, and Jeroen Van De Graaf. "Towards a Publicly-Verifiable Mix-Net Providing". In: *Financial Cryptography and Data Security. FC 2013. Lecture Notes in Computer Science* 7859 (2013), pp. 197–204. DOI: 10.1007/978-3-642-39884-1_16.

[BG13] Stephanie Bayer and Jens Groth. "Zero-knowledge argument for polynomial evaluation with application to blacklists". In: *Lecture Notes in Computer Science* 7881 LNCS (2013), pp. 646–663. ISSN: 03029743. DOI: 10.1007/978-3-642-38348-9_38.

[BGG19] Sean Bowe, Ariel Gabizon, and Matthew D. Green. "A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK". In: *Lecture Notes in Computer Science* 10958 LNCS (2019), pp. 64–77. ISSN: 16113349. DOI: 10.1007/978-3-662-58820-8_5.

[BKM19] Ethan Buchman, Jae Kwon, and Zarko Milosevic. *The latest gossip on BFT consensus.* 2019. arXiv: 1807.04938.

[BMGK+07] Kevin Bauer et al. "Low-resource routing attacks against Tor". In: *WPES'07 - Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society* (2007), pp. 11–20. DOI: 10.1145/1314333.1314336.

[BP15] Alex Biryukov and Ivan Pustogarov. “Bitcoin over Tor isn’t a good idea”. In: *2015 IEEE Symposium on Security and Privacy*. San Jose, CA, 2015, pp. 122–134. ISBN: 9781467369497. DOI: `10.1109/SP.2015.15`. arXiv: `1410.6079`.

[BRRS+15] Josh Benaloh et al. *End-to-end verifiability*. 2015. arXiv: `1504.03778`.

[Bun13] Schweizerische Bundeskanzlei. *Verordnung der BK über die elektronische Stimmabgabe (VEleS)*. 2013. URL: `https://www.admin.ch/opc/de/classified-compilation/20132343/index.html` (visited on 04/09/2020).

[CGJ17] Andrea Caforio, Linus Gasser, and Philipp Jovanovic. “A Decentralized and Distributed E-voting Scheme Based on Verifiable Cryptographic Shuffles”. 2017. URL: `https://www.epfl.ch/labs/dedis/wp-content/uploads/2020/01/report-2017-2-andrea_caforio-evoting.pdf`.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. “A secure and optimally efficient multi-authority election scheme”. In: *Lecture Notes in Computer Science* 1233 (1997), pp. 103–118. ISSN: 16113349. DOI: `10.1007/3-540-69053_9`.

[Cha88] David Chaum. “The dining cryptographers problem: Unconditional sender and recipient untraceability”. In: *Journal of Cryptology* 1 (1988), pp. 65–75. ISSN: 09332790. DOI: `10.1007/BF00206326`.

[Com05] Estonian National Electoral Committee. *E-Voting System General Overview*. Tallinn, 2005.

[CPP13] Édouard Cuvelier, Olivier Pereira, and Thomas Peters. “Election verifiability or ballot privacy: Do we need to choose?” In: *Lecture Notes in Computer Science* 8134 LNCS (2013), pp. 481–498. ISSN: 03029743. DOI: `10.1007/978-3-642-40203-6_27`.

[CV17] Christian Cachin and Marko Vukolić. “Blockchain consensus protocols in the wild”. In: *Leibniz International Proceedings in Informatics, LIPIcs* 91 (2017). ISSN: 18688969. DOI: `10.4230/LIPIcs.DISC.2017.1`. arXiv: `1707.01873`.

[Dam02] Ivan Damgard. *On Σ-protocols*. 2002. URL: `https://cs.au.dk/~ivan/Sigma.pdf`.

[DGA12] Denise Demirel, J Van De Graaf, and R Araújo. “Improving Helios with Everlasting Privacy Towards the Public”. In: *EVT/WOTE’12 Proceedings of the 2012 international conference on Electronic Voting Technology/Workshop on Trustworthy Elections* (2012). URL: `https://www.usenix.org/system/files/conference/evtwote12/evtwote12-final13.pdf`.

[E-V] E-Voting.CC GmbH. *World Map of Electronic Voting*. URL: `https://www.e-voting.cc/en/it-elections/world-map/` (visited on 04/20/2020).

[Eur16] European Parliament and The Council of the EU. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016*. 2016. URL: `https://gdpr-info.eu/` (visited on 04/11/2020).

[FS87] Amos Fiat and Adi Shamir. “How to prove yourself: Practical solutions to identification and signature problems”. In: *Lecture Notes in Computer Science* 263 LNCS (1987), pp. 186–194. ISSN: 16113349. DOI: `10.1007/3-540-47721-7_12`.

[GB20] Damien Giry and BlueKrypt. *Cryptographic Key Length Recommendation.* 2020. URL: `https://www.keylength.com/` (visited on 04/20/2020).

[GKTP16] J. Paul Gibson et al. "A review of E-voting: the past, present and future". In: *Annales des Telecommunications/Annals of Telecommunications* 71.7-8 (2016), pp. 279–286. ISSN: 19589395. DOI: `10.1007/s12243-016-0525-8`.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "Knowledge complexity of interactive proof systems". In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208. ISSN: 00975397. DOI: `10.1137/0218012`.

[Gro04] Jens Groth. "Efficient maximal privacy in boardroom voting and anonymous broadcast". In: *Lecture Notes in Computer Science* 3110 (2004), pp. 90–104. ISSN: 03029743. DOI: `10.1007/978-3-540-27809-2_10`.

[HABS+16] Ethan Heilman et al. *TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub.* Cryptology ePrint Archive, Report 2016/575. `https://eprint.iacr.org/2016/575`. 2016.

[Hae13] Rolf Haenni. *UniVote System Specification.* 2013. URL: `https://e-voting.bfh.ch/app/download/5874743461/specification.pdf?t=1507600656` (visited on 04/15/2020).

[HH16] Severin Hauser and Rolf Haenni. "Implementing Broadcast Channels with Memory for Electronic Voting Systems". In: *JeDEM - eJournal of eDemocracy and Open Government* 8.3 (2016), pp. 61–79. ISSN: 2075-9517. DOI: `10.29379/jedem.v8i3.441`.

[HHA18] Ryan Henry, Amir Herzberg, and Kate Aniket. "Blockchain Access Privacy: Challenges and Directions". In: *IEEE Security & Privacy* 16 (2018), pp. 38–45. DOI: `10.1109/MSP.2018.3111245`.

[HKLD17] Rolf Haenni et al. *CHVote System Specification.* Cryptology ePrint Archive, Report 2017/325. `https://eprint.iacr.org/2017/325`. 2017.

[HKLD18] Rolf Haenni et al. *CHVote System Specification Version 1.4.1.* Biel, CH, 2018.

[HL09] James Heather and David Lundin. "The append-Only web bulletin board". In: *Lecture Notes in Computer Science* 5491 LNCS (2009), pp. 242–256. ISSN: 03029743. DOI: `10.1007/978-3-642-01465-9_16`.

[HLG20] Rolf Haenni, Philipp Locher, and Nicolas Gailly. "Improving the Performance of Cryptographic Voting Protocols". In: *Financial Cryptography and Data Security. FC 2019. Lecture Notes in Computer Science* 11599 (2020). ISSN: 16113349. DOI: `10.1007/978-3-030-43725-1_19`.

[HPW15] Sven Heiberg, Arnis Parsovs, and Jan Willemson. "E-Voting and Identity". In: *E-Voting and Identity. Vote-ID 2015. Lecture Notes in Computer Science* 9269 (2015), pp. 19–34. ISSN: 16113349. DOI: `10.1007/978-3-319-22270-7`.

[HS11] Rolf Haenni and Oliver Spycher. "Secure Internet Voting on Limited Devices with Anonymized DSA Public Keys". In: *Proceedings of the 2011 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections* (2011), p. 8. URL: `http://dl.acm.org/citation.cfm?id=2028012.2028020`.

[JCJ10] Ari Juels, Dario Catalano, and Markus Jakobsson. "Coercion-resistant electronic elections". In: *Towards Trustworthy Elections. Lecture Notes in*

Computer Science, vol 6000 (2010), pp. 61–70. DOI: 10.1007/978-3-642-12980-3_2.

[JMP13] Hugo Jonker, Sjouke Mauw, and Jun Pang. "Privacy and verifiability in voting systems: Methods, developments and trends". In: *Computer Science Review* 10 (2013), pp. 1–30. ISSN: 15740137. DOI: 10.1016/j.cosrev.2013.08.002.

[JN06] Hugo Jonker and Open Universiteit Nederland. "Formalising Receipt-Freeness". In: *Information Security. ISC 2006. Lecture Notes in Computer Science.* Vol. 4176. Springer, Berlin, Heidelberg, 2006. DOI: 10.1007/11836810_34.

[KDKV+18] R. Krimmer et al. "How Much Does an e-Vote Cost? Cost Comparison per Vote in Multichannel Elections in Estonia". In: *Electronic Voting. E-Vote-ID 2018. Lecture Notes in Computer Science* 11143 (2018), pp. 117–131. DOI: 10.1007/978-3-030-00419-4_8.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography.* CRC PRESS Boca, 2007.

[Kri19] Robert Krimmer. "A Structure for New Voting Technologies: What They Are, How They Are Used and Why". In: The Art of Structuring. Springer, Cham, 2019, pp. 421–426. ISBN: 978-3-030-06233-0. DOI: 10.1007/978-3-030-06234-7_39.

[KRMS+20] Christian Killer et al. "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.* SAC '20. Brno, Czech Republic: Association for Computing Machinery, 2020, pp. 286–293. ISBN: 9781450368667. DOI: 10.1145/3341105.3373884.

[KRS10] Steve Kremer, Mark Ryan, and Ben Smyth. "Election verifiability in electronic voting protocols". In: *ESORICS'10, 15th European Symposium on Research in Computer Security* 6345 LNCS (2010), pp. 389–404. ISSN: 03029743. DOI: 10.1007/978-3-642-15497-3_24.

[KTV15] Oksana Kulyk, Vanessa Teague, and Melanie Volkamer. "Extending Helios Towards Private Eligibility Verifiability". In: *E-Voting and Identity. Vote-ID 2015.* 9269 (2015), pp. 57–73. ISSN: 16113349. DOI: 10.1007/978-3-319-22270-7.

[KY02] Aggelos Kiayias and Moti Yung. "Self-tallying Elections and Perfect Ballot Secrecy". In: *PKC'02, 5th Interna- tional Workshop on Theory and Practice in Public Key Cryptography, LNCS* 2274 (2002), pp. 141–158. ISSN: 16113349. DOI: 10.1007/3-540-45664-3_10.

[Lan05] Serge Lang. *Undergraduate Algebra.* Springer, New York, NY, 2005. ISBN: 978-0-387-27475-1. DOI: 10.1007/0-387-27475-8.

[Len04] Arjen K. Lenstra. "Key Lengths. Contribution to The Handbook of Information Security". 2004.

[LH15] Philipp Locher and Rolf Haenni. "Verifiable internet elections with everlasting privacy and minimal trust". In: *Lecture Notes in Computer Science* 9269 (2015), pp. 74–91. ISSN: 16113349. DOI: 10.1007/978-3-319-22270-7_5.

[Lin] Linux Foundation. *Official Hyperledger Fabric Documentation.* URL: https://hyperledger-fabric.readthedocs.io/en/latest/ (visited on 04/15/2020).

[Loc16] Philipp E. Locher. “Unconditional Privacy in Remote Electronic Voting Theory and Practice”. Doctoral Thesis. University of Fribourg, 2016.

[Maa04] Epp Maaten. “Towards remote e-voting: Estonian case”. In: *Electronic Voting in Europe – Technology, Law, Politics and Society*. Bregenz, AT, 2004, pp. 83–90.

[Mau19] A. Driza Maurer. “The Swiss Post/Scytl Transparency Exercise and Its Possible Impact on Internet Voting Regulation”. In: *Electronic Voting. E-Vote-ID 2019. Lecture Notes in Computer Science* 11759 (2019), pp. 83–99. DOI: `10.1007/978-3-030-30625-0_6`.

[MN06] Tal Moran and Moni Naor. “Receipt-free universally-verifiable voting with everlasting privacy”. In: *Advances in Cryptology - CRYPTO 2006* LNCS 4117 (2006), pp. 373–392. ISSN: 16113349. DOI: `10.1007/11818175_22`.

[Nat13] National Institute of Standards and Technology. *FIPS PUB 186-4: Digital Signature Standard (DSS)*. Gaithersburg, MD, 2013. URL: `https://oag.ca.gov/sites/all/files/agweb/pdfs/erds1/fips_pub_07_2013.pdf`.

[Neo] Neo Foundation. *Official Neo Documentation*. URL: `https://docs.neo.org/` (visited on 04/15/2020).

[NKJG+17] Kirill Nikitin et al. “CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds”. In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1271–1287. ISBN: 978-1-931971-40-9. URL: `https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/nikitin`.

[OSC13] OSCE/ODIHR. *Handbook For the Observation of New Voting Technologies*. OSCE Office for Democratic Institutions and Human Rights (ODIHR), 2013. ISBN: 9789292348694. URL: `http://www.osce.org/odihr/elections/104939?download=true`.

[Par] Parity. *Consensus in Substrate*. URL: `https://substrate.dev/docs/en/conceptual/core/consensus#consensus-in-substrate` (visited on 04/15/2020).

[Ped92] Torben Pryds Pedersen. “Non-interactive and information-theoretic secure verifiable secret sharing”. In: *Lecture Notes in Computer Science* 576 LNCS (1992), pp. 129–140. ISSN: 16113349. DOI: `10.1007/3-540-46766-1_9`.

[Pos19] Swiss Post. *Ballot box not hacked, errors in the source code - Swiss Post temporarily suspends its e-voting system*. 2019. URL: `https://www.post.ch/en/about-us/media/press-releases/2019/swiss-post-temporarily-suspends-its-e-voting-system` (visited on 04/09/2020).

[PT19] Olivier Pereira and Vanessa Teague. *Report on the SwissPost-Scytl e-voting system , trusted-server version*. 2019.

[RMK17] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. “P2P Mixing and Unlinkable Bitcoin Transactions”. In: *NDSS Symposium 2017*. San Diego, California, 2017. DOI: `10.14722/ndss.2017.23415`.

[SBV17] João Sousa, Alysson Bessani, and Marko Vukolic. “A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform”. In: *CoRR* abs/1709.06921 (2017). arXiv: `1709.06921`.

[Sch89] C.P. Schnorr. “Efficient Identification and Signatures for Smart Cards”. In: *Advances in Cryptology — CRYPTO' 89 Proceedings. CRYPTO 1989. Lecture Notes in Computer Science* 435 (1989), pp. 239–252. DOI: `10.1007/0-387-34805-0_22`.

[SFDK+14] Drew Springall et al. “Security analysis of the estonian internet voting system”. In: *Proceedings of the ACM Conference on Computer and Communications Security* (2014), pp. 703–715. ISSN: 15437221. DOI: `10.1145/2660267.2660315`.

[SGMP+15] Uwe Serdult et al. “Fifteen years of internet voting in Switzerland: History, Governance and Use”. In: *2015 2nd International Conference on eDemocracy and eGovernment, ICEDEG 2015* (2015), pp. 126–132. DOI: `10.1109/ICEDEG.2015.7114482`.

[SK95] Kazue Sako and Joe Kilian. “Receipt-free mix-type voting scheme - A practical solution to the implementation of a voting booth”. In: *Advances in Cryptology, (EUROCRYPT 1995).* Vol. 921. Springer, Berlin Heidelberg, 1995, pp. 393–403. ISBN: 3540594094. DOI: `10.1007/3-540-49264-X_32`.

[Sma16] Nigel P. Smart. *Cryptography Made Simple.* Springer International Publishing, 2016. ISBN: 978-3-319-21935-6. DOI: `10.1007/978-3-319-21936-3`.

[Szi17] Péter Szilágyi. *Github Issue for Clique PoA protocol.* 2017. URL: `https://github.com/ethereum/EIPs/issues/225` (visited on 04/15/2020).

[Ten] Tendermint Inc. *Official Tendermint Core Documentation.* URL: `https://docs.tendermint.com/` (visited on 04/15/2020).

[Vin12] Priit Vinkel. “Internet voting in Estonia”. In: *Information Security Technology for Applications. NordSec 2011. Lecture Notes in Computer Science* 7161 (2012), pp. 4–12. DOI: `978-3-642-29615-4_2`.

[WHHM+19] Wenbo Wang et al. “A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks”. In: *EEE Access* 7 (2019). DOI: `10.1109/ACCESS.2019.2896108`. arXiv: `arXiv:1805.02707v4`.

[Woo14] Gavin Wood. *Ethereum: a secure decentralised generalised transaction ledger.* 2014. URL: `http://gavwood.com/paper.pdf` (visited on 04/14/2020).

[ZXDC+17] Zibin Zheng et al. “An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends”. In: *Proceedings - 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017* (2017), pp. 557–564. DOI: `10.1109/BigDataCongress.2017.85`.

Abbreviations

BFT	Byzantine Fault Tolerance
BP	Ballot Privacy
CLI	Command Line Interface
CRS	Common Reference String
DC-Net	Dining Cryptographers Network
DDH	Decisional Diffie-Hellman
DL	Discrete Logarithm
DoS	Denial of Service
IdM	Identity Management
NIST	National Institute of Standards and Technology
PBB	Public Bulletin Board
PoA	Proof-of-Authority
PoS	Proof-of-Stake
PoW	Proof-of-Work
REST	Representational State Transfer
REV	Remote Electronic Voting
RPC	Remote Procedure Call
TPS	Transactions Per Second
ZKP	Zero-Knowledge Proof

Glossary

Mix-Net A Mix-Net in the context of electronic voting is a system that takes votes as inputs and shuffles them such that no element in the output can be linked back to an element of the input. If the votes were connected to voter identities before shuffling, the mix-net outputs the votes with privacy established.

Homomorphic Encryption Homomorphic Encryption allows computation on the ciphertext. Results obtained from operations performed on the ciphertext can be decrypted and match the result as if the operation would have been performed on the plaintext. This is interesting in electronic voting, where we want to keep ballots encrypted but still accumulate them and have access to the vote count in plaintext.

Distributed Key Generation Distributed Key Generation is a cryptographic process in which multiple parties contribute to the calculation of a key pair in which every party possesses a part of the private key. To make use of the private key, *e.g.,* to decrypt a value encrypted with the public key, the parties have to cooperate.

Secret Sharing Secret Sharing refers to methods for distributing a private key amongst a group of participants, each of whom is only in possession of a part of the key. The whole key can be reconstructed only when a minimum number of participants cooperate.

Computational Intractability An intractable problem in computer science is one that is assumed to be computationally hard or infeasible with bounded computational power. It is possible that such a problem becomes feasible in the future because of new methods of solving it or increased computational power.

Dining Cryptographers Network A Dining Cryptographers Network is a network that allows its participants to send anonymous messages. The anonymity in such a network is unconditional, meaning it does not depend on computational intractability.

Byzantine Fault Tolerance A system with Byzantine Fault Tolerance (BFT) is robust against byzantine faults. If this is applied to distributed systems, it means that up to one third of network participants can become faulty or even malicious without threatening the system's functionality. In the blockchain context, BFT is a desirable property of consensus mechanisms.

Blockchain Validator A Validator in a blockchain network is a node that takes part in the consensus mechanism and thereby validates incoming transactions. It is synonymous with a consensus node.

Blockchain Oracle An Oracle in blockchain is a data source outside of the blockchain network. Their data can be used in transactions on the blockchain.

Public Bulletin Board A Public Bulletin Board (PBB) is an append-only broadcast channel from which anyone can read, but write access might be controlled. The data on the PBB must not be modifiable, or modification must be at least detectable.

Voting Authority A Voting Authority is an entity responsible for the setup and execution of an election or poll. In this work, they are the equivalent of a voting administrator.

Administration Client The Administration Client is the application that allows the voting authorities to configure and manage an election.

Voter Client The Voter Client is the application that allows the voter to cast votes from her electronic devices.

Public Credential In the UP protocol the public credential is a number generated by the voter and registered under her identity on the PBB. It is generated from the private credentials.

Private Credentials In the UP protocol the private credentials are two numbers generated by the voter and used to generate the public credential and the election credential. They are kept secret.

Election Credential In the UP protocol the election credentials is a number generated by the voter and included in the ballot. It is generated from part of the private credentials.

Voting Period The Voting Period is the time of the election in which voters can cast ballots, *i.e.,* the system is open to receive ballots.

Secure Channel A Secure Channel is a communication channel between two parties from that an observer can only see encrypted messages being exchanged. The observer can see the individual messages, but they must be computationally secure.

Private Channel A Private Channel is a communication channel between two parties from that an observer can only see encrypted messages being exchanged. The observer can see the individual messages, but they must be information-theoretically secure.

Untappable Channel An Untappable Channel is a communication channel between two parties from that an observer cannot observe anything but the fact that two parties are communicating. The observer does not see individual messages (encrypted or not) being exchanged.

Anonymous Channel An Anonymous Channel is a communication channel which makes it impossible for the receiver of a message to determine who the sender is. More generally, it makes it impossible for any observer to link the sender and receiver of a message.

JSON-RPC JSON-RPC is a simple protocol for remote procedure calls encoded in JSON.

List of Figures

List of Tables

Appendix A

Installation and Usage

A.1 Installation Guide

The source code for the electronic voting system implemented in this thesis can be found at `https://github.com/csmuller/up-voting-system`.

To install the applications, Golang in version 1.13.0 or higher is required. If Go is not yet installed on your system, consult the official Go installation guide at `https://golang.org/doc/install` or use your favorite package manager (*e.g.,* Homebrew) to install it.

Make sure that the environment variable `GOPATH` is set[1] and the directory at `GOPATH/bin` is set in your `PATH` variable. The prototype binaries will be installed into that directory later.

Clone the repository

```
mkdir -p $GOPATH/src/github.com/csmuller
cd $GOPATH/src/github.com/csmuller
git clone https://github.com/csmuller/up-voting-system.git
```

If you get the code from somewhere else, make sure to put it into the same directory structure as shown above, *i.e.,* at `$GOPATH/src/github.com/csmuller`.

The PBB, voter client and administration client applications are installed using the Makefile included in the repository. The following command installs all three application binaries (`pbbd`, `vcli`, and `acli`) into `GOPATH/bin`.

```
cd up-voting-system
make install
```

Now you should be able to run the following commands. A new shell session might be required to make the commands available.

[1] `https://github.com/golang/go/wiki/GOPATH`

```
pbbd help
vcli help
acli help
```

A.2 Usage Guide

To test the prototype locally with a single PBB node do the following.

Initialize the bulletin board node and change some of the nodes configurations. This creates the directory `~/.pbbd` in your home directory and sets up some basic configuration files. It then changes a few configurations concerning the transactions sizes to make sure that transactions are not rejected because of their size.

```
pbbd init val --chain-id pbb
sed -i -e 's/^max_body_bytes.*/max_body_bytes = 10000000/' \
 ~/.pbbd/config/config.toml
sed -i -e 's/^max_packet_msg_payload_size.*/max_packet_msg_payload_size = 10000000/' \
 ~/.pbbd/config/config.toml
sed -i -e 's/^max_tx_bytes.*/max_tx_bytes = 10000000/' ~/.pbbd/config/config.toml
```

Next, the client applications are invoked for creation of new accounts (key pairs) used to sign transactions. Note that the password used for theses accounts is "12345678". Calling the client applications for the first time creates the directories `~/.vcli` and `~/.acli` in your home directory. The generated key pairs are stored in those directories.

```
echo "12345678" | acli keys add admin
echo "12345678" | vcli keys add voter
```

The clients can be configure for more convenient use.

```
acli config chain-id pbb
acli config output json
acli config indent true
acli config trust-node true

vcli config chain-id pbb
vcli config output json
vcli config indent true
vcli config trust-node true
```

Next, the administration account is added as a genesis account to the genesis block. It is needed in the next command to sign a genesis transaction. Note that the tokens attributed to the account (*i.e.,* stake and foo) can have any name. Any token denoted here will later exist on the blockchain. Though, the `stake` token has a special functionality in the PoS consensus mechanism that is run by the Cosmos-SDK. When the account is used to create a validator, its `stake` tokens become bound as stake. Since the PBB doesn't offer any monetary value the `stake` doesn't have any value, but it is still needed for the system to function.

```
pbbd add-genesis-account $(acli keys show admin -a) 100000000stake,1000foo
```

With the admins account in the genesis block, a genesis transaction can be added that promotes the local PBB node to the status of a validator in the consensus mechanism. The second command collects the generated transaction and applies it to the genesis block. Genesis transactions are places by default in the directory `~/.pbbd/config/gentx`. Likewise the `collect-gentxs` command collects them from the same directory.

```
echo "12345678" | pbbd gentx --details val --name admin
pbbd collect-gentxs
```

The PBB node is now ready to be started with the following command.

```
pbbd start
```

To check if the PBB is working correctly, a first transaction from the admin to the voter account can be performed. The admin account was set up with 1000 `foo` tokens and can transfer those to the voter account. First, retrieve the voter account's address.

```
vcli keys show voter -a
```

Use this address in the next command by replacing the place holder address that starts with 'cosmos'.

```
echo "12345678" | acli tx bank send $(acli keys show admin -a) \
cosmos1j7dzx2cd78raru4rp3m7pf7xm6dl7g5u3j7rs3 1foo -y
```

The following command shows if the account now exists on the PBB.

```
vcli query account $(vcli keys show voter -a)
```

Note, that the protocol parameters are already on the PBB. They are encoded in the source code and can be retrieved with the following command. Without a file path as an argument the retrieved parameters are saved to the `~/.vcli` folder.

```
vcli query pbb params
```

The voter can now create private and public credentials and register the public part on the PBB. The command requires three arguments. The first two are the location where the generated public and private credentials should be stored and the third is the file containing the parameters that we fetched before. The `-from` flag denotes the account to use for signing the transaction.

```
vcli tx pbb new-voter ~/.vcli/cred.pub ~/.vcli/cred.priv ~/.vcli/params.json \
    --from $(vcli keys show voter -a)
```

Because the prototype does not check voter identities, we can create multiple new voters with the same key pair (the 'voter' account in this case). I.e., the above command can be repeated to register multiple voters, each with newly generated public and private credentials. The command will override the credential files on your disk every time it is called.

The voter credential(s) should now show up in the list of public credentials is queried with the next command. Additionally, the credential polynomial should be up to date. The polynomial command requires a file argument where it will be stored in. It is later required in the command that creates ballot transactions.

```
vcli query pbb credentials
vcli query pbb poly ~/.vcli/poly.json
```

After registering a couple of voters, we can start to cast ballots. The `vote` command takes at least three arguments. The first is a string for the vote, the second and third are the locations of the public and private credentials. There are two non-mandatory arguments that denote the location of the polynomial and protocol parameters. By default they are looked for at `~/.vcli/poly.json` and `~/.vcli/params.json`, respectively.

```
vcli tx pbb vote yes ~/.vcli/cred.pub ~/.vcli/cred.priv \
    --from $(vcli keys show voter -a) \
    --gas 1000000000000
```

The `-gas` flag is needed because Cosmos-SDK uses the concept of gas for transaction processing similar to Ethereum. Although, the concept is of no use in the PBB, it needs to be handled as part of Cosmos-SDK. Setting the `-gas` flag to a very high value prevents transactions from failing because of a low default gas maximum setting.

The voter account was again used for signing the transaction. This is simply done for convenience but would actually remove ballot privacy because we used the same account for registering the voter. In a real-world scenario the account used here should be a publicly available key pair that is used by every voter while the account used for the registration is specific to every voter.

Finally, after casting a ballot the vote-tallying can be performed. The command used for this fetches all ballots from the PBB, verifies the contained ZKP transcripts, and writes the votes of the valid ballots into a file at `~/.vcli/votes.txt`. Because the votes can have any format, no accumulation is attempted automatically.

```
vcli query pbb verify
```

Troubleshooting

If at some point transactions seem to be successfully build and send but do not show up on the bulletin board, the problem might be that the transactions hit the default gas limit. Note, that the PBB does not make use of the Gas notion but still needs to deal with it as an integral part of Cosmos-SDK. To solve this problem add the `-gas 1000000000000` flag to the transaction command, as in the following example.

```
vcli tx pbb new-voter ~/.vcli/cred.pub ~/.vcli/cred.priv \
    ~/.vcli$h/params.json \
    --from $(vcli keys show voter -a) \
    --gas 1000000000000
```

Appendix B

Contents of the CD

This work comes with a CD containing the following items:

- The source code of the implemented prototype and command line scripts for operating it.
- A PDF file of this report.
- The source files of this report (incl. graphics).
- The data used to produce the evaluation Figures 6.1 and 6.2.
- The code used to produce the evaluation Figures 6.1 and 6.2.