University of Zurich[UZH]

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

Cast-as-Intended Verifiability in Blockchain-based Electronic Voting for Swiss National Elections

Raphael Matile
Zurich, Switzerland
Student ID: 12-711-222

Supervisor: Bruno Rodrigues, Eder Scheid
Date of Submission: November 12, 2018

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

ifi

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

Zusammenfassung

Die Transformation der Demokratie in das digitale Zeitalter beschäftigt die Forschung seit geraumer Zeit. Ihre zentrale Stütze, die politische Partizipation des Elektorats in wiederkehrenden Abstimmungen, ist da keine Ausnahme. Die Kryptographie erforscht daher verschiedenste Ansätze zur sicheren elektronischen Abstimmung. Jedoch ist das in den Allgemeinen Menschenrechten garantierte Recht auf vertrauliche Wahlen keine Selbstverständlichkeit und auf dem elektronischen Weg nicht einfach umzusetzen. Weitere Bedürfnisse, wie die universelle Verifizierbarkeit einer jeden abgegebenen Stimme, sind oft von gegensätzlicher Natur. Nicht nur mit dem gestiegenen Misstrauen gegenüber freien und sicheren Abstimmungen durch die aufgetauchten Unregelmässigkeiten während den letzten präsidialen Wahlen in den Vereinigten Staaten von Amerika ist es von zentraler Bedeutung, die Integrität des Abstimmungsresultats sicherzustellen. Verteilte Systeme und ihre Konsensfindung haben daher mit dem enstandenen öffentlichen Interesse an Blockchains ein neues Forschungsgebiet gefunden: die Zusammenführung von elektronischen Abstimmungen auf verteilten Systemen. Das föderalistische politische System der Schweiz bildet eine solche dezentralisierte Topologie und eignet sich daher hervorragend für ein elektronisches Abstimmungssystem, das seine Aufgaben über mehrere Autoritäten verteilt. Zusammen können Kantone sowie Gemeinden ein dezentralisiertes Netzwerk bilden, auf welchem die Abstimmungssoftware ausgeführt wird. Eine zentrale Entität wird so nicht benötigt. Daher schlägt diese Arbeit zum ersten Mal ein Blockchain-basiertes elektronisches Abstimmungssystem vor, welches jedem Stimmberechtigten erlaubt, die verschlüsselte Repräsentation seiner Stimme mithilfe eines kenntnisfreien Beweises auf ihre Gültigkeit zu verifizieren, ohne gleichzeitig das Wahlgeheminis zu beeinträchtigen. Durch das lineare Verhalten der dazu benötigten kryptografischen Verfahren eignet sich das vorgestellte Protokoll auch für nationale Wahlen. Jedoch ist es noch nicht vollständig Ende-zu-Ende verifizierbar. Protokollerweiterungen können die dazu erforderlichen Schritte jederzeit nachliefern.

Abstract

Democracy in the digital age has attracted a lot of public attention in recent years. Its fundamental principle of participating in electoral processes is not an exception. However, transforming analogue procedures to their digital counterparts often require specific concepts, such as cryptography, on which enabling technologies are built. Indeed, cryptographic research has a long history of designing secure electronic voting systems. However, bringing the human right of secrecy in voting to electronic systems is difficult. Other properties, such as the possibility of verifying universally that any vote counted was indeed the decision made by a voter, are often conflicting and a trade-off must be found. With the recent distrust in free and secure elections in the United States of America, it has become even more important to guarantee the integrity of any election. With the raise of public interest in blockchains, research around distributed systems and consensus algorithms have found a new field of application: Bringing secure electronic voting to decentralised systems. Switzerland with its federal political structure is a perfect fit for implementing an electronic voting system where trust is distributed among multiple authorities. Together, cantons and even municipalities can build a decentralised network running the election software, avoiding a central entity which needs to be trusted. Thus, this thesis proposes the first blockchain-based electronic voting system providing cast-as-intended verifiability. By using a non-interactive zero-knowledge proof of knowledge, any voter can verify that his or her encrypted vote represents the chosen decision while still maintaining the secrecy of the ballot. In addition, any required cryptographic material can be generated in linear time to the number of voters, making the outlined system suitable even for large scale elections. As the presented prototype is not yet fully end-to-end verifiable, extensions to the current protocol can provide these features in the future.

Acknowledgments

Many people have supported me emotionally and personally during the course of this thesis. Special thanks goes to Bruno Rodrigues, my supervisor, for his valuable thoughts and suggestions; Thomas Bocek and Christian Killer, for all their inputs and discussions on electronic voting systems along the way; Andreas Gruhler, for his corrections; and Nathalie, for her endless encouragement.

Contents

Chapter 1

Introduction

Participating in votes and elections is a phenomenon which has been around for many centuries. However, different means have been used to cast a vote. From spoken word, raising hands, papyrus or paper, a multitude of representations for a ballot have been known and used. In the last two decades, also electronic ways for casting a vote have been evaluated and introduced in electoral processes. Estonia has seen legally-binding electronic voting since 2005, and also Switzerland has been experimenting with electronic voting channels since the beginning of the 21st century. Although some suggest electronic voting increases voter turnout, others state only insignificant changes have been observed. With electronic voting channels, questions of trust arise when submitting intangible ballots. Democracies with their checks and balances may already provide trusted means of voting. However, the recent presidential election of the United States of America in 2016 has shown that also in long living democracies trust issues in elections occur [41]. In other countries, transparent electronic voting systems are even considered the only means of conducting trustful elections [38]. Instead of relying on the honesty of one administering governmental entity, decentralised structures with independent implementations of a particular process cannot only distribute the risk of being attacked successfully, but might also increase the immediacy of a voter's participation in the election. Starting with the publication of Bitcoin [51], a peer-to-peer electronic cash system, in 2008, decentralised systems and distributed consensus algorithms have seen a revived public interest. Its fundamental data structure, an append-only chain of blocks, shows great potential to many researchers and investors.

1.1 Motivation and Description of Work

Recently, there have been attempts to implement electronic voting on top of such a decentralised structure, removing the need to trust a single authority. Although well-known properties exist in the research literature for evaluating pivotal requirements, few works aim at designing approaches which consider them. In particular, coercion-resistance and receipt-freeness, ballot secrecy, and verifiability of each submitted vote must be taken into account [8, 43]. In 2013, Switzerland has released new regulations which define the

requirements any electronic voting system must met in order to be certified for hosting legally-binding elections [25]. With Switzerland's federal structure, one fundamental requirement for decentralised electronic voting is already recognised: Distributing parts of the voting process to multiple authorities. Thus, establishing a similar structure for an electronic voting system seems to be the only logical consequence. By allowing the municipalities and cantons providing part of the electronic infrastructure, a decentralised peer-to-peer network could be created. Thus, this work is twofold: First, current available approaches to cast-as-intended verifiability are researched, compared, and evaluated with respect to a decentralised electronic voting system. Second, the best fitting approach is implemented by using a single-purpose blockchain as its fundamental data structure.

1.2 Thesis Outline

In cryptography, much research has been conducted which targets electronic voting. Consequentially, different approaches have been undertaken to introduce electronic voting systems worldwide. A short overview of them is provided at the beginning of Chapter 2. It further outlines some properties any electronic voting system should consider before presenting an overview of cryptographic fundamentals on how to achieve them. This chapter concludes with approaches to previous blockchain-based electronic voting systems. In Chapter 3, the architecture of a blockchain-based implementation for an electronic voting system is discussed. Chapter 4 then discusses the implementation-specific details which are evaluated in Chapter 5. A summary of the work achieved and suggested improvements is given in Chapter 6.

Chapter 2

Background and Related Work

Participating in direct democratic processes from remote places was known already back in the Roman Empire, either by raising the voice or clapping swords [59]. With the raise of reliable postal services, votes could be transported to their place of destination [34]. Although used in Switzerland until today, submitting votes over an electronic channel also has experienced a rise in the last years. Electronic voting is known in various distinct kinds but is sometimes shaped into new forms by combining a set of unique features. In [52], four types of electronic voting are classified: (1) Ballot Scanning Technology which attempts to scan physical ballot papers directly at a polling station; (2) Direct Recording Electronic (DRE) Voting Systems that describe electronic devices at polling stations on which a voter can directly input his choice; (3) Internet Voting which allows to vote in uncontrolled environments at nearly any place on the world; and (4) Hybrid forms which use the centralised counting mechanism of Internet voting but combine it with the controlled environment of polling stations.

2.1 Electronic Voting Landscape

Electronic voting is a relatively new practical field. From 1996 to 2007, only a total of 136 elections used a form of remote electronic voting [47]. Since then, advances in technology have lead to further trials and binding votes, however.

2.1.1 Electronic Voting around the World

In Europe, efforts for introducing electronic voting have succeeded in few countries. Estonia is one example. Based on their previously established usage of a digital identity called SmartID, Estonian citizens can submit votes over the Internet [58]. Also in the United States of America, multiple attempts have been made. Although electronic voting is widely used by means of Email and Fax [42], the National Institute of Standardization and Technology (NIST) has suggested that further research and development is required to make voting over the Internet feasible [39]. In many developing countries, interest in

Country	n	%	Country	n	%	Country	n	%	Country	n	%
Nigeria	19	27.54	Iran	3	4.35	Tanzania	2	2.90	Mauritius	1	1.45
Brazil	7	10.14	Jordan	3	4.35	UAE	2	2.90	Mexico	1	1.45
Indonesia	5	7.25	S. Africa	3	4.35	Ghana	2	2.90	Thailand	1	1.45
Argentine	4	5.80	Colombia	2	2.90	Ecuador	1	1.45	Turkey	1	1.45
India	3	4.35	Pakistan	2	2.90	Lebanon	1	1.45	Uganda	1	1.45
									Others	5	7.25

Table 2.1: Distribution of English research articles with respect to electronic voting in developing countries [38]

experimenting with electronic voting systems is present. In 2016, [38] has performed a literature review, counting the number of scientific articles written in English and published with respect to electronic voting on a country basis. The results are shown in Table 2.1. Based on this, the authors analyse a common theme among the articles suggesting that credible elections can only made possible by technology.

2.1.2 Electronic Voting in Switzerland

In Switzerland, multiple experiments on electronic voting systems have been performed as well. In 2000, a project evaluating opportunities, risks, and the feasibility of electronic voting has been initiated on request of parliamentary motions [24]. In this context, the cantons of Geneva, Neuchâtel and Zurich attempted to provide their citizens with implementations of electronic voting [23]. Although allowed initially only for canton-wide elections, all three systems managed to perform nationwide votes as soon as 2004 (Geneva) and 2005 (Neuchâtel and Zurich). Based on this success, the disjoint consortium Vote êlectronique was founded in 2009 with seven cantons [24]. Four years later, in 2013, new federal regulations [25] introduced thresholds up to how many percent of the electorate is allowed to vote electronically. These thresholds are bound to a set of requirements an electronic voting system has to meet in order to be admitted for cantonal and nationwide elections. Based on these requirements, the Federal Council of Switzerland decided not to admit the project of the consortium for a nationwide vote. Following this decision, the consortium was disbanded [57]. Subsequently, the canton of Neuâtel has joined his forces with The Swiss Post in developing an electronic voting system since both were using an implementation of the Spanish vendor Scytl. Only the implementation of the canton of Geneva has also seen continued development. This situation remains until today.

2.2 System Properties

According to [43], the following properties should be considered for private and verifiable elections performed on electronic voting systems. Besides of listing constitutional requirements of electronic voting systems and their consequences as done in [43], [36] also outlines user requirements and use-cases. A less common attribute is described as *accessibility*, a property primarily focused by the end-users of an electronic voting system. It aims at

providing an indiscriminate way of submitting votes for a wide range of users with highly different characteristics. [3] has identified further attributes a voting system should fulfill: Among others, *soundness* of a voting system describes its ability to guarantee the absence of faulty processes and illegitimate operations. The location-independent participation of a voter in the election is further addressed in the property of *mobility* [3].

2.2.1 Ballot Secrecy

A requirement for establishing electronic voting in the context of democratic election processes is secrecy[1] in voting. Importantly, this property is also stated in Article 21 of the Universal Declaration of Human Rights [62]. Related to ballot secrecy is the ability of having a free-choice during the vote. In [55], it is formalised as "Ballot secrecy: A voter's vote is not revealed to anyone". With establishing ballot secrecy in an electronic voting system, new challenges emerge which must be coped with. Submitting a vote in an uncontrolled environment (as opposed to a polling station) allows a voter being observed and its choices made public. In addition, systems creating a link between a vote and a voter should ensure that decryption is not considered safe due to assumptions in the cryptographic algorithms, as such constraints can change with future developments [8]. This property is known as everlasting privacy [50] or unconditional privacy [17].

2.2.2 Coercion-Resistance and Receipt-Freeness

Related to ballot secrecy are the properties of coercion-resistance and receipt-freeness. The first is fulfilled for a voting system "if there exists a way for a coerced voter to cast her vote such that her coercer cannot distinguish whether or not she followed the coercer's instructions" [8]. Hence, it is crucial to prevent forced abstention, where a voter is hindered from submitting his choice; forced surrender of credentials by which a coercer can obtain the secret credentials to participant in a vote as the voter; and forced randomisation, in which the coercer dictates the voter to choose always the same choice. Attempts to solve one of these constraints often fail with respect to the two others. Instead, coercion evidence can be obtained by systems allowing to submit multiple votes with the same credentials [8].

Receipt-freeness is closely related to coercion-resistance: A voting system leaking information on what options voters have selected provides a channel of useful information to the adversary. Thus, receipt-free systems ensure that: "a voter is unable to prove how she voted even if she actively colludes with a coercer and deviates from the protocol in order to try to produce a proof" [8].

[1]The term ballot privacy and ballot secrecy is often used interchangeably. However, to be consistent with the notion in [55], we adopt its reasoning to "avoid confusion with other privacy notions, such as receipt-freeness and coercion resistance" [55] and prefer ballot secrecy over ballot privacy.

2.2.3 Verifiability

Once a vote is cast it must be ensured that it still represents the actual choice made by a voter, i.e. whether the vote was casted as intended. As opposed to paper ballot voting, electronic devices may alter votes invisibly to the end-user or they even might execute a different protocol from the one expected. This assurance is often referred to as verifiability. Whereas [46] differentiates between individual, universal and eligibility verifiability, end-to-end verifiability is also mentioned in the literature [43].

Individual Verifiability According to the definition from [46], electronic voting systems providing individual verifiability ensure that "a voter can check that her own ballot is included in the election's bulletin board."

Universal Verifiability Widening that view to further parties other than the voter herself, such as election observers, universal verifiability is outlined by [46] as the possibility that "anyone can check that the election outcome corresponds to the ballots published on the bulletin board."

Eligibility verifiability Electronic voting systems further need to ensure the eligibility of submitted votes, a feature referred to from the authors of [46] as "anyone can check that each vote in the election outcome was cast by a registered voter and there is at most one vote per voter."

End-to-End Verifiability Whereas individual, universal and eligibility verifiability centre around systems trying to provide a high degree of privacy while still being verifiable, end-to-end verifiability focuses on practical verifiability while maintaining strong privacy notions. End-to-end verifiability is fulfilled if voters can verify the three attributes [43]:

- CAST-AS-INTENDED: "her choice was correctly denoted on the ballot by the system"
- RECORDED-AS-CAST: "her ballot was received the way she cast it"
- TALLIED-AS-RECORDED or COUNTED-AS-RECORDED: "her ballot counts as received"

In the research literature, these verifiability notions have seen attempts by which they are formalised, such as [17, 46].

2.2.4 Incompatibility of Properties

In [17], a formal definition of some of the above properties is outlined. In particular, these are unconditional privacy, receipt-freeness, and universal verifiability. Based on this formalism, they state two incompatibilities as Theorem 5 and Theorem 6 of their work, which are depicted in the following excerpts:

Theorem 5. *"In the standard model, it is impossible to build a voting scheme that simultaneously achieves the* universal verifiability *and the* unconditional privacy *unless all the voters actually vote."* [17]

Theorem 6. *"If there exists a function h such that $B_i = h(P, V_i, v_i, r_i)$, where v_i is the vote of the voter V_i, r_i a possibly random value chosen by V_i, and P some public information, then the universal verifiability and the receipt-freeness properties cannot be simultaneously achieved without additional assumptions."* [17]

Thus, the goal of any voting system is to find a suitable trade-off between simultaneously providing privacy and verifiability [43].

2.2.5 Approaches to Cast-as-Intended Verifiability

Voting Codes One approach followed by Pretty Good Democracy (PGD), consists of requiring users typing voting codes to choose a particular voting option [54]. After the vote is submitted, an acknowledgement code must be verified for the submitted choice.

Return Codes and Plaintext Equivalent Tests Similarly to voting codes, return codes allow for cast-as-intended verifiability. However, return codes do not require to specify codes for choosing a voting option but are usually verified after the chosen values have been sent to a voting server.

An improvement of the Norwegian voting protocol outlined in [33] allows to type in a plain representation of the given voting options. As the chosen voting options have been received by the voting server, return codes are generated on a further system and returned to the voting server. The codes are then forwarded to the client device. By using a previously obtained confirmation value, the voter indicates that the retrieved return codes match the ones previously retrieved: A procedure on the client device generates a confirmation message which is again sent to the voting server, where a finalisation code is generated and returned to the voting device. This code has again to be verified with a previously specified value by the voter.

A similar approach is outlined in [37]: Oblivious Transfer is used as its underlying cryptographic protocol. In the simplest variant, this protocol consists of two parties, a sender and a receiver. The sender holds a defined set of messages of which some can be selected by the receiver. It is noteworthy, that by using this protocol, the sender will not know which messages the receiver has selected while the receiver has no knowledge about the values of the non-selected messages. The protocol is initiated by having the receiver querying the sender for a particular message. In turn, a response is delivered from the sender to the receiver which can be opened by the receiver. As in the return code scheme, a verification code sheet is generated on a voter-basis using the hash values of a number of points $P_{i,s}$ of a particular polynomial p_i, with $s = 1, 2, ..., n$ voting options. During the vote casting process, a query is initiated by the voting platform to retrieve corresponding values which open to the points P_i of the polynomial of the voter. By computing the hash of

the retrieved points and comparing the resulting values with the initially obtained verification codes, the voters can ensure that their votes were cast as intended.

However, this approach was to be found as being flawed in [12]. The same authors also published a different approach of using return codes than [33]: Although this protocol also includes the phases of verifying return codes and typing in verification codes, the verification process on the voting server is different. It utilises Plaintext-Equivalent-Tests (PETs) which represent a zero-knowledge protocol for verifying that two ciphertexts hold the same plaintext value. In the particular instance presented, one ciphertext is submitted by the voter whereas the other is reconstructed by values subject to the verification codes generated initially.

Universal Cast-as-Intended Verifiability The previously outlined approaches are all highly interactive and require a voter willing and able to participate in all phases of the voting process. The authors of [31] therefore propose a single-pass voting scheme, characterised by a single interaction of the voter with the voting system, the submission of the ballot. As before, codes tied to a specific voting option for a particular user are generated during registration. These values are hashed and a subset of them are used during the vote casting step to form a Non-Interactive Zero-Knowledge Proof of Knowledge (NIZKPK). Such a proof is not only verifiable by the voters themselves but also allow for public verification.

Assumptions

All of the approaches assume explicitly [12, 33, 37] or implicitly [31] a public append-only data structure to which the voting authorities can write particular cryptographic material. Further, some of the approaches [33, 37] allow only for one vote to be submitted by the same voter, whereas the others do not limit themselves to this restriction. [31, 33, 37] further explicitly state that the list of voting options must be predefined, whereas [12] assumes this implicitly. To provide cast-as-intended verifiability, [33] and [37] support single-voting only, where a voter can only submit his choices once during the election. In the work of [31], the party distributing the appropriate cryptographic material needs to be honest. Similar, [12] states that the printing facility of the voting cards is trusted.

Comparative Criteria

Table 2.2 outlines a brief overview of the different approaches. Most of the protocols require more than one interaction with the voting system to allow for cast-as-intended verification (*Interactive*). In addition, these protocols may be bound to a specific approach of how votes must be processed. Only [37] does not require a specific underlying mechanism to tally votes but postpones a concrete integration to a specific implementation. [12] even presents corresponding security proofs which they claim are independent of the voting mechanism used. Further, some of the approaches directly include a method to validate the votes for their appropriate message domain during processing. The security attributes *coercion-resistance* and *receipt-freeness* are not focused in the outlined papers

Protocol	Interactive	Supports	Limitations
Return Codes [33]	Yes	Shuffling [33]	No duplicates, Blank Vote, Synchronization [18]
NIZKPK [31]	No	Mix-nets, Homomorphic Tallying [31]	–
Return Codes on OT [37]	Yes	n/a [37]	Invalid Votes [12]
Return Codes with PETs [12]	Yes	Mix-nets, Homomorphic Tallying	–

Table 2.2: Cast-as-intended mechanisms in comparison

and are thus omitted as comparison criteria. Two of the mentioned approaches have security issues: For the work presented in [33], a security analysis has been performed in [18]. Among others, the authors noticed an issue where a voter has to be prevented of voting twice for the same voting option k_i (of n total ones). In addition, a usability concern was raised, which arises when a voter has to submit different return codes for an abstained voting option. The presented protocol further looses its cast-as-intended property as soon as a voter submits a vote twice to the voting system, which could occur when multiple instances of the server must be deployed. In addition, [12] described an attack on [37] where an adversary can submit invalid votes which are only detected once they have been decrypted (and the link to the ballot and voter is lost).

2.3 Cryptographic Fundamentals

The field of cryptographic research has found applications in various different technologies. As such, it often has been an enabler for providing data integrity and privacy. The resulting cryptographic tools are also widely integrated in electronic voting systems to ensure secure and private elections. Where data is encrypted and operations are performed on such representations, new means of verifying its opaque contents need to be applied. In this section, a short introduction into the cryptographic primitives used in this thesis is given.

2.3.1 Finite Cyclic Groups

In the following, most arithmetic operations are calculated over a finite, cyclic group instead of just the set of all integers $\mathbb{Z}$. As cyclic groups and the operations on them may be unfamiliar to the reader, a short excerpt of necessary algebraic facts is given. Where not stated differently, these are taken from [45].

Definition 1 - Group.

A *group* is a pair $(G, \star)$, where G is a set and $\star$ is a binary operation over G, such that the following axioms hold.

1. The binary operation is associative. $\forall a, b, c \in G | a \star (b \star c) = (a \star b) \star c$.
2. There is an element $e \in G$ such that for all $a \in G$ we have $e \star a = a \star e = a$, i.e. the neutral element e of $(G, \star)$.
3. For all $x \in G$, there exists a $y \in G$ (x^{-1}) such that $x \star y = y \star x = e$, i.e. the inverse element y.

Definition 2 - Finite Cyclic Group.

A group G is called *cyclic* if there exists a $g \in G$, such that for all $h \in G$, there exists an $x \in \mathbb{Z}$ such that $g^x = h$. In this case, g is called the *generator* of G, and we write $G = \langle g \rangle$. In addition, one can state that "only finite cyclic groups are of the form $\mathbb{Z}_n$ for some integer n". [45]

2.3.2 Homomorphic Encryption in the ElGamal Cryptosystem

Often, a cryptosystem is only considered cryptographically secure with assumptions on the computational complexity of certain operations. The ElGamal cryptosystem [30] explicitly relies on the computational hardness of finding the discrete logarithm of g^x for large numbers x in finite fields, such as the finite cyclic groups outlined above.

Some cryptosystems further provide the property of *homomorphism*, which is a mapping between two algebras of the same type, whereas the type can be a cyclic group, among others. With respect to electronic voting, such a homomorphism can be applied by using a cryptosystem, mapping operations on plaintext votes $\oplus$ to operations on encrypted votes $\otimes$ [43]. Consider two plaintext messages m_1, m_2 of the set of all messages M, then the homomorphic operation is defined as:

$$\forall m_1, m_2 \in M, \;\; enc(m_1 \oplus m_2) = enc(m_1) \otimes enc(m_2)$$

Based on this relation, one can define the operation $\oplus$ as multiplication and $\otimes$ as the addition over the corresponding type, i.e. the finite cyclic group. This allows to calculate the final tally on encrypted votes, without the need of decrypting each ciphertext to its plain text vote.

The public-key cryptosystem of ElGamal provides such an additive homomorphism. With having the fundamental computation assumption in mind, encryption and decryption can be performed as outlined in Definition 3. Although ElGamal is originally created with a multiplicative homomorphism, homomorphic addition can also be performed, as specified in Definition 4.

Definition 3 - Encryption and Decryption.

Define the message space of all valid plain-text votes to be in the cyclic subgroup G of order q of $(\mathbb{Z}_p)^*$, with q being co-prime to p and g being the generator of G. Then,

1. Create a *private key* by selecting a random number x from the uniformly distributed set $\{1, ..., q-1\}$ and keep x secret.

2. Create the corresponding *public key*, by calculating $h = g^x$. Make the set $\big(G, q, g, h\big)$ public.

3. Encrypt a message $m \in \mathbb{Z}_p$ using $r \in_{\text{uniform}} \mathbb{Z}_p$ with the public key h by calculating the shared secret $s = h^r = (g^x)^r = g^{xr}$. Then, the resulting ciphertext is defined as $E(G, H) = \big(g^r, g^m \cdot s\big)$.

4. Decrypt a ciphertext $E(G, H)$, by recalculating the secret $s = G^x = (g^r)^x = g^{rx}$ and $g^m = H \cdot (s^{-1}) = g^m \cdot h^r \cdot (g^{xr})^{-1} = g^m \cdot g^{xr} \cdot g^{-xr}$ with s^{-1} being the modular multiplicative inverse of s. Then, solve the discrete logarithm in order to obtain m.

Definition 4 - Homomorphic Addition.

Then, having obtained two ElGamal ciphertexts $E(m_1)$ and $E(m_2)$, the homomorphic addition is defined as follows:

$$\begin{aligned} E(m_1) \cdot E(m_2) &= E(G_1, H_1) \cdot E(G_2, H_2) \\ &= E(g^{r_1}, g^{m_1} \cdot h^{r_1}) \cdot E(g^{r_2}, g^{m_2} \cdot h^{r_2}) \\ &= E(g^{r_1+r_2}, g^{m_1+m_2} \cdot h^{r_1+r_2}) \\ &= E(m_1 + m_2) \end{aligned}$$

2.3.3 Non-Interactive Zero-Knowledge Proofs

Encrypted representations are a source of distrust to voters. Just by visibly inspecting such a sequence of characters, a voter will not be satisfied that his vote is still accurately represented. In addition, when using homomorphic encryption, votes are mapped into the domain of the cyclic group. As such, election authorities cannot be ensured to count only votes of the appropriate range: Consider an election with a single binary voting option. Then, a voter could not only encrypt valid votes, i.e. yes or no which are mapped into the domain values of 1 and 0, respectively, but also any other numeric value, such as 22. Thus, proofs for valid votes are required. By using a cryptographic tool called non-interactive zero-knowledge proof, it is possible to verify such properties without revealing anything about the actual choice a voter has made [10]. Whereas Definition 5 is due to [35], Definition 6 is taken from [10] and Definition 7 from [45].

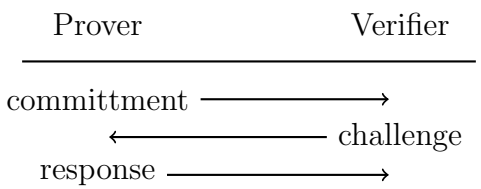


Figure 2.1: The three moves of a Σ-protocol

Definition 5 - Interactive Proof System.

Let L be a language over $(0,1)^*$. Let (A, B) be an interactive protocol with the two Interactive Turing machines (ITM) A and B. We say that (A, B) is an *interactive proof system* for L if we have the following:

1. For each k, for sufficiently large x in L given as input to (A, B), B halts and accepts with probability at least $1 - |x|^{-k}$. (The probabilities here are taken over the coin tosses of A and B.)

2. For each k, for sufficiently large x not in L, for any ITM A', on input x to (A', B), B accepts with probability at max $|x|^{-k}$. (The probabilities here are taken over the coin tosses of A' and B.)

Remark: The above probability for error can be decreased, say to smaller than $2^{-|x|}$, by the standard technique of repeating the protocol many times and choosing to accept by majority vote.

Such interactive proofs can be combined with the property of zero-knowledge:

Definition 6 - Zero-Knowledge.

Zero-knowledge guarantees that the proof gives no knowledge, but the validity of the theorem.

In short, an interactive proof system can be seen as "games played between two players, Prover and Verifier, who can talk back and forth" [10]. In the above definition, the two parties are represented by the interactive Turing machines A and B. In the following, only a subset of such interactive proofs will be necessary. In particular, these are *three-move-protocols*, often referred to as Σ-protocols [21], which have the exchange of information between the two parties, verifier and prover as outlined in Figure 2.1. Such interactive protocols can be made non-interactive by using the Fiat-Shamir heuristic [32]. The following illustration is taken from [45]: If a prover wishes to prove the validity of a statement x, it first generates a commitment y and a challenge $c = H(y, x)$ whereas H is defined as a suitable cryptographic hash function. It further generates a response s based on the commitment y and challenge s created before.

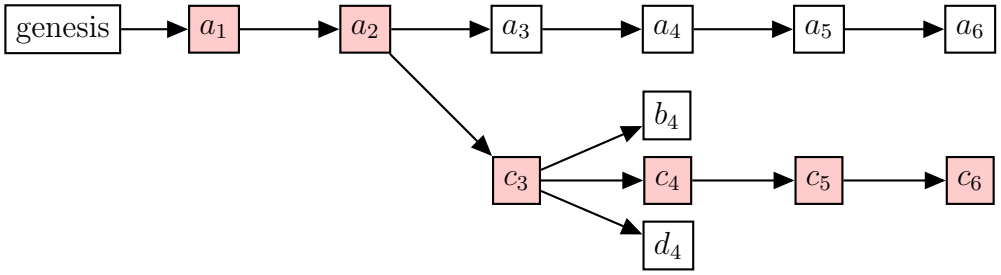


Figure 2.2: A chain of blocks with its canonical chain marked in red

Then, the triple (y, c, s) is representing a static proof of the validity of x. In other words, the hash function H ensures that the challenge is computed after the commitment has been chosen. The security of this proof is based on the Random Oracle Model (ROM) which assumes that the hash function H behaves as a random oracle [45].

Definition 7 - Random Oracle.

A random oracle is a deterministic function whose output is uniformly distributed in its range.

Using these cryptographic primitives one can construct proofs ensuring a vote is valid and represent the choice of the voter without the necessity of revealing the plaintext vote. These are outlined in more detail in Chapter 3.

2.4 Blockchain

Where trust in an intermediary authority is at stake, transparency in its workings is key. In elections, voters need to trust the established processes and their government acting as such intermediary. Instead of trusting such an intermediary, a distributed system could be used to store votes in a transparent and verifiable way. By appending votes to a distributed and nearly immutable append-only data structure, a common source of trust can be built [11]. As blockchains accumulate transactions into blocks and append them to a distributed view of truth, a chain of information is built. Such a chain is depicted in Figure 2.2 whereas the common agreed view of information is built by the changes stored in the blocks marked in red. As its name suggests, a blockchain is a sequence of blocks, each holding a collection of transactions. Among other fields, a block always holds a reference to its parent. Transactions in turn contain the actual payload by which the common agreed truth is built [64].

2.4.1 Consensus Algorithms

In distributed systems, a number of independent and distributed parties work together to achieve a common goal. Connections between these can fail, i.e. messages can be discarded, corrupted or new messages may be delivered. This arbitrary behaviour is often

generalised under the term of *Byzantine Failures* [19]. The resulting problem of finding an agreement in distributed systems has been recognised already back in 1982 and was described in terms of the *Byzantine Generals Problem* [48].

Blockchains as an epitome of distributed systems naturally suffer from the same issues. In [64], a comparison of different blockchain architectures and their consensus algorithms has been presented. The authors distinguished between *permissionless* blockchains which allow everyone to participate in the consensus process, and *permissioned* blockchains which restrict write access to a well-known committee of nodes. Membership is granted based on defined policies and its formation can be either static with a non-changing set of members, or dynamic where committee members are added and removed [6]. Based on the participation allowance, these types of blockchains are often called *open* and *closed*, respectively. Due to resulting architectural properties, both types follow different approaches to reach consensus. For permissionless blockchains, *Proof-of-Work* (PoW), *Proof-of-Stake* (PoS), *Delegated Proof-of-Stake* (DPoS) and *Ripple* know concrete implementations, among others. Permissioned blockchains use, non-exhaustively, *Practical Byzantine Fault Tolerance* (PBFT), *Proof-of-Authority* (PoA) or *Tendermint* as their consensus algorithm [22]. A more extensive overview of blockchains with their supported consensus algorithms is provided in [11]. [4] compared blockchains with respect to various performance attributes.

In [33], blockchains are described to be a trusted system. As such, they have to ensure *dependability*, a term subsuming non-exhaustively *reliability*, *availability*, *safety*, and *integrity* [5]. Thus, [33] further states that "blockchains replicate data only for resilience, not for scalability". In [6], consensus protocols are considered in terms of (1) *liveness* which is subdivided into *validity* and *agreement*, and (2) *safety* with its sub-dimensions *integrity* and *total order*. Both, (1) and (2), refer to *atomic broadcast*, a concept describing a sequence of two asynchronous events. These are *broadcast* and *deliver*, whereas the first is invoked by a node attempting to broadcast a message, and the latter is invoked by the broadcast protocol delivering the message to a local application running on the node. Then, validity ensures that if a node is broadcasting a message, it is also delivered by the node. Agreement states that if a message is delivered by a correct node, the same message is eventually delivered by every correct node. Integrity further guarantees only-once delivery for a particular message and therefore allows for deriving that the sender of a message has previously broadcast the message. Eventually, total order refers to all correct nodes extracting the same order of messages. *Crash-tolerant consensus* protocols provide these properties while maintaining a threshold of crashed nodes whereas in *byzantine consensus* nodes actively working against the common goal are tolerated [15] . As such, transactions cannot be considered to be valid in general but require validation.

Further, the work of [6] refers to consensus as being an "*'agreement' by all nodes, not 'choice'*". As such, consensus is not considered a voting protocol due to an adversary being able to control the order of messages sent to the system.

Permissionless Consensus

In PoW, a computationally hard problem has to be solved by any participant in the network to allow for incorporating transactions into the blockchain. Instead of working

constantly on such a computationally hard problem, PoS allows network participants to incorporate transactions in proportion to a specified type of stake. Whereas PoS is reflecting a direct democracy for its consensus, DPoS uses elected representatives. Due to the reduced number of nodes which have to confirm a transaction, an improved confirmation time is promised. In Ripple, *server* subnetworks decide whether a transaction should be incorporated into the blockchain. Contrary, *client* nodes only transfer funds. Whenever a transaction needs to be confirmed, a server node asks a set of others which can agree or deny the transaction. If more than 80% agree, the transaction is approved [64].

Permissioned Consensus

The family of Byzantine Fault Tolerant (BFT) algorithms follows a different approach to reach consensus. By using a messaging scheme, network nodes attempt to agree on incorporating a transaction into the blockchain. To protect the network against nodes publishing malicious transactions, a voting mechanism is often put into place, allowing for removal of such nodes.

Practical Byzantine Fault Tolerance (PBFT) describes a message schema requiring $3f+1$ node replicas, for f faulty nodes. It operates as follows: A client sends an operation on a replicated state machine to its primary node. This, in turn, forwards the operation to its replicas. Eventually, replicas send the result of the operation back to the client which acknowledges as result the value of $f+1$ same responses [16].

Proof-of-Authority (PoA) is claimed to provide performance improvements over general BFT protocols due to its lighter message exchange. In a PoA scheme, n trusted nodes are allowed to mine blocks in a round-robin manner, from which $\frac{n}{2}+1$ need to be honest. To elect a node as a primary which is allowed to mine a single block, time is divided into steps. Then, each node can calculate whether its his turn to mine at each given point in time [22].

Tendermint requires three consecutive steps (forming a round) in order to agree on a new block: pre-vote, pre-commit, and commit. In each, a $\frac{2}{3}$ majority of nodes has to be found which agree on the problem statement. In each round of following these steps, *validator* nodes start by deciding whether they submit a pre-vote for a proposed block. If a node receives the necessary majority of pre-votes stated above, it broadcasts a pre-commit message. Again, if the node has received a majority of such messages, it validates the block and publishes a commit message. Eventually, if the majority of commit messages is reached, a node accepts the block [13].

Avalanche is a recent proposal of a BFT algorithm which utilizes confidence counters to decide on the validity of a transaction. Nodes build up these counters by querying the confidence values of a transaction from a fixed amount of other peers. A transaction will receive a so-called chit, if it is the preferred transaction of the queried node and all its ancestor transactions are preferred as well. In case a chit is received as result from the query, the confidence value of a transaction is increased. Consensus is achieved by defining a threshold of confidence by which a transaction is said to be agreed on [53].

Consensus in Voting

Voting in a democratic setting requires central coordination, at least up to the extent of agreeing on the voting question, the voting period, and the decision of a voter's eligibility. Thus, permissionless consensus does not provide much value. Permissioned consensus, however, allows for capitalising on a countries democratic structure. Often, the role of a voting authority can be split up among a set of different electoral parties, such as counties, cantons or even municipalities. Similarly, a scenario where representatives of political parties cooperate to administer a voting is possible.

2.5 Related Work

As of today, The Swiss Post distributes its solution for electronic voting with an implementation based on return codes. Votes can be submitted by prior authorisation to the voting server. Based on the received ballot, a set of return codes is generated and transmitted to the voter. Then, the return codes must be verified by the voter before submitting a confirmation message to the voting server again. Subsequently, a finalisation code is generated, stored and sent back to the voter. A vote is considered to be successfully received if the finalisation code matches an equivalent on the printed voting card [33]. Its competitor, the canton of Geneva, uses a similar end-user process but a different implementation. Before votes are decrypted and tallied, they are mixed and partially decrypted by a set of voting authorities. These partially decrypted values are then retrieved by an election administrator and converted back to plain-text values. Then, these values are summed up to calculate the final tally [37].

Whereas multiple approaches have been developed to perform electronic voting on conventional systems, only few practical implementations are available which combine voting with the realm of distributed ledgers. In 2017, [49] has presented a feasibility study of a boardroom voting solution based on a smart contract running on the Ethereum blockchain. It represents the public bulletin board where all required election information is stored. Votes are encrypted prior to sending them to the Ethereum network. Further, a 1-out-of-2 zero-knowledge proof ensures that the encrypted vote contains either a zero or one vote. The entire voting process includes a setup, sign-up, commit (optional), vote, and tally phase, of which the sign-up, commit, and vote phase have to be performed by the voter. Participants have to announce their private voting key during the sign-up phase and then commit to their chosen voting option by publishing a hash of their encrypted vote. Eventually, they send the encrypted vote and the zero-knowledge proof in a transaction to the smart contract. As stated in their work, the major drawback of the presented solution roots in the self-tallying voting protocol used, which allows the last voter to abort the entire election. The authors cope with this issue by providing a financial incentive for each voter when they complete the entire voting process. As each voter has access to the public bulletin board, they claim to provide recorded-as-cast and cast-as-intended verifiability. Counted-as-record should be fulfilled as each party can recompute the tallied result. However, there is no formal proof given, supporting the stated verifiability claims.

In the same year, [63] has proposed an electronic voting solutions based on ring signatures on the Bitcoin blockchain. After a two-step registration phase, the voter is signed up for participating in the election. During voting, each participant has to retrieve all public keys of all other voters which he will use in combination with its own private key in order to sign the chosen election option. A commitment is then sent to the Bitcoin address of the election authority. To tally, the voting authorities collect all retrieved transactions and verify their signature. If they are valid, the corresponding counter for a particular candidate option is increased. The author claims to have fulfilled individual verifiability, but no formal proof is provided.

In spring 2018, Agora [1] has performed a trial voting on their blockchain-based implementation of an electronic voting system for the presidential elections in Sierra Leone. As registered election observers, members of Agora were allowed to manually register votes in 280 polling stations [1, 2].

Chapter 3

Architecture

This work attempts to allow voters to take political action in an uncontrolled environment, such as the Internet. Therefore, it is best categorised as Internet Voting within the classification of New Voting Technology introduced by [52]. As such, the system requires at least two bodies operating different components in any election[1]: *Voting authorities* which operate a component to manage, store and tally votes and code executed on the *Voters*' end-clients, such as a computer or mobile device.

In the Swiss federation, cantons and municipalities are independent bodies [26]. Hence, they implement the right to vote independently and according to their constitution [27]. Considering this legal structure for an electronic voting system, independence of these bodies with a simultaneous cooperation for the common goal of providing secure elections to their citizens can be ensured. Thus, the system to design should capitalise on this structure and involve the different bodies in providing key components of the infrastructure.

This chapter is structured as follows: First, general considerations with respect to the security of a voting systems are outlined. Then, requirements based on these considerations are noted. Section 3.4 elaborates on assumptions which narrow the scope of this work.

3.1 Design Considerations

A large set of attack vectors exists for electronic voting systems, not only with respect to the network operating the storage layer and the tallying process but also on the client devices running the end-user software. This thesis focuses primarily on verifying that the cast vote is indeed representing the voter's choice. Thus, a brief outline of some attack vectors affecting this security property in particular is given in the following.

[1] Although the word election and vote are semantically different, in the former meaning electing representatives and the latter meaning voting on some (binary) question, they are often used interchangeably among the literature. In the following, this difference is relaxed as well, letting both terms refer to a vote on a specific question.

3.1.1 Secure Connections

Connections between participants of the voting system's communication network should be secure. In [43], *sender-anonymous channels* and *untappable communication* channels are outlined as requirements for a secure voting process. Whereas the first guarantees that a sender cannot be identified by the receiver of a message, the latter ensures that no party but the sender and receiver "can learn anything about the communication, including whether communication occurred or not" [43].

3.1.2 Malicious Client

In order to alter the result of the final tally, the client software which allows a voter to submit his decisions is naturally a targeted victim. Attacks against voting clients would have a high impact on the voting outcome. Thus, it is of great significance that this risk can be minimised. There are two places where the software for the submission of the vote could run. On the one side, the client software could be hosted on the controlled infrastructure and provided by the voting authorities. On the other side, software could be distributed to the end-users and directly executed on their own devices. Both approaches have their advantages and shortcomings and offer different attack vectors.

Consider the latter case, where a malicious client participates in the voting process obtaining the cryptographic credentials of the user who runs the client application of the election software. To place a successful attack, the adversary would require to modify the client software binary so that it may alter the typed in decision as well as adjusting any validation checks put into place. Additionally, the attacker would ideally hide any attempt of tampering with the end-user's vote. Such an application running directly on an end-user client can be subject to integrity protection which verifies the application's integrity before executing it. This can include different strategies, such as white-listing or black-listing applications or using digital signatures for the application's binaries. In fact, these strategies find their applications in Apple's Gatekeeper[2].

The former setting, in which the voting authorities provide the client software to the user, suffers from an additional attack vector. Not only the integrity of the software executed by the end-user must be ensured but also the communication between the end-user and the infrastructure provided by the voting authorities. Web-applications as a way of providing the client software to the voters, pose a special risk for tampering with users' data as a browser's execution can be influenced by installed extensions. This is also recognised in work published by Scytl, the company developing The Swiss Post's e-voting system [20]. Scytl outlines two possibilities to verify the integrity of the browser application's code: (1) Signing the code using public key cryptography, which is impractical due to missing standards and (2) utilising the *W3C Subresource Integrity*[3] recommendation to verify the integrity of JavaScript code linked in the HTML document rendered by the browser, which is not yet fully implemented by all browser vendors and does not allow verifying the integrity of the file in which the sub-resource's fingerprints are defined.

[2] `https://support.apple.com/en-us/HT202491`
[3] `https://www.w3.org/TR/SRI/`

Despite these limitations, trust is put into their voting clients [20]: On the one side, secure connections to the browser are enforced when loading the web application's code. On the other hand, integrity checks on the server distributing the source files are executed. In addition, a dedicated JavaScript application downloads source code as a regular HTTP client and compares the received code with respect to a prior generated baseline. In their conclusion, a trade-off in favour of a less integer but more interoperable system with better user-experience is made [20]:

> *"In comparison to Java implementations, the overall security of Javascript voting clients is similar, whereas the user experience and interoperability is largely better since these clients are much lighter and multi-platform than the Java ones. A browser with Javascript support is the only usage requirement. The fact that there is no dependency with the Java Runtime Environment (JRE) has extensively reduced the usability and interoperability problems, as well as the exposure to critical security bugs, associated to the former Java implementations. Regarding the security, the only disadvantage is the Javascript implementation lacks the support for signing the code. However, additional security measures mitigate this issue, e.g. remote code integrity validation services and use of verifiable voting protocols allowing the voter to verify the vote cast with independence of the voting client logics."*

3.1.3 Malicious Network Participants

In any service offering public interfaces, it is key that their consumers follow the protocol rules for interaction. As interacting clients are often not controlled by the entity providing these interfaces, enforcing valid requests is hard. However, requiring only integer clients is considered too restrictive. Instead, public interfaces must be able to handle inappropriate or even invalid requests.

Transaction Censorship

When submitting transactions to a single network node only, the node itself could pretend to not have received the vote at all, eventually filtering unwanted decisions from being considered in the final tally. Therefore, the voting system would not consider the new vote. However, the incentive for such a behaviour is low: If the vote is encrypted and the voter's identity is not disclosed to the node receiving the vote (e.g. by using masked identifications), it's strategy would result in arbitrarily dropping requests and therefore not being able to affect the final tally in any particular outcome.

Identity Spoofing

A more significant issue is encountered when adversarial nodes attempt to announce arbitrary votes to the voting system. As clients submit information about their identity which allows for verifying their eligibility, nodes could also use the set of eligible voter identities to announce transactions on behalf of them. By following the standard procedure to announce transactions to voting system, they may not distinguish where the vote initially originates from.

3.1.4 Tallying in Switzerland's Federal System

Although an established democracy guarantees honesty of its executive bodies to some extent, considering a malicious player is still important. To counteract attacks on elections performed electronically, the Swiss law requires to consider many properties used in the research literature. Among others, [25] states that a risk assessment must be performed for *ballot secrecy*, *coercion-resistance*, and *receipt-freeness*. If further defines that for targeting up to 50% of the cantonal electorate, *cast-as-intended*, and *cast-as-recorded* verifiability must be guaranteed. Any electronic voting system involving more than half of the cantonal electorate must further ensure *counted-as-recorded* verifiability [25].

In Switzerland, the tallying process is within the responsibility of the cantons for national elections [27]. Often, voting registers are maintained on a cantonal basis. As this caused major issues in trial elections [9], such a federalistic topology should be considered during a requirements elicitation phase.

3.2 Requirements

As discussed in the literature for electronic voting and further also defined as a basic requirement for electronic voting in Switzerland, it is essential that the voting system ensures verifiability of a vote.

Cast-as-intended, recorded-as-cast, and counted-as-recorded verifiability must be guaranteed by the voting system. (R1)

Submitting a vote privately is essential to account for Article 21 in the Universal Declaration of Human Rights.

Ballot secrecy should be guaranteed. (R2)

The voting protocol must allow voters to participate in an election without having to perform a complex registration and setup phase prior to submitting their votes.

The steps of the voting protocol should therefore be comparable to the ones a voter has to accomplish for paper-based voting. (R3)

In addition, the voting protocol must ensure that voting is free of charge to allow voting without financial preliminaries. As pre-funding any kind of account is an additional administrative burden, it should be avoided.

Approaches requiring pre-funding of a voter account should be avoided. (R4)

3.3 System Topology

Voting is an act of trust in the electoral processes. Client-server architectures often imply that a single authority operates and maintains a particular system. In addition, access to storage layers is rarely allowed to an external party. Observing elections within such a setup relies on having transparent system providers in place which grant access to crucial system components. In contrast, open peer-to-peer networks require any participating party to behave according to a common protocol and therefore need a certain amount of coordination. However, in such a network each partaker is generally allowed to have the same view on the network's information. Changing the system's topology from a client-server to a peer-to-peer network may be a critical source of transparency for the electronic projection of the electoral processes: Not only the single authority providing the infrastructure of the system would be required to be malicious but an orchestrated set of participants.

As political interests often span many distinct affairs and are scattered among multiple parties representing one view or another, they would form an appropriate distributed system provider with a common interest: Performing secure and valid electronic elections. Coordination between them, however, could be a tedious task: As there are various topics, it might be hard to determine which interest groups should be involved in providing the entire system's infrastructure. Thus, one could approach this issue by involving all individuals of an electorate in providing the infrastructure together. Although more people are experienced in using software today than ever, not all might be willing or able to setup a part of the voting system by themselves just for submitting a single choice.

By using already established but decentralised structures for providing the required infrastructure, the workload of setting up the electronic voting system can be taken away from the individual voter or party. Having such structures represented by the federalistic members of the democratic system in Switzerland (i.e. municipalities within cantons,

cantons within the confederation), these would form an ideal network of providers of the voting system. And yet they would still be able to perform independent votes on a cantonal or nationwide level. Therefore, a peer-to-peer network in which each canton or municipality would run a single peer, represents an appropriate fit for providing the voting infrastructure.

Maintaining durable connections between nodes in a distributed system is not a straightforward task: Nodes can disconnect from the peer-to-peer network at any time due to many reasons. Therefore, using a protocol for cast-as-intended verifiability, which requires only a few interactions, is preferable. Indeed, the protocol outlined in [31] allows to submit a vote in a single step and is thus considered a good fit for this problem statement.

3.4 Assumptions

Prior assumptions can impact any systems architecture as they affect which design decisions are derived [14]. Documenting them prior to specify a particular system and thus making them explicit is key. In the following, central premises are outlined, taking into account some of the legal requirements of Switzerland.

In any election performed in Switzerland, every eligible participant is restricted to cast only a single vote [28]. This contrasts from approaches to coercion-resistant systems which only count the last submitted decision [8].

Every single voter is allowed to cast a single vote only. (A1)

To ensure A1, every voter needs to be uniquely identified. This does, however, not necessarily imply public verifiable links between voters and their votes.

Every person of the electorate is uniquely identified. (A2)

As the number of Swiss cantons and municipalities change rarely, the authorities of an election are assumed to be known and fixed in their representation. Therefore, a peer-to-peer network operating the system's infrastructure does not need to take into account a procedure in which participants can be added or removed.

The nodes in the peer-to-peer network providing the election system's infrastructure are fixed and known prior to performing the vote. (A3)

When votes are submitted from an end-user device, such as a browser or native application, connections to the voting system must be confidential.

The connection between an end-user's voting device and the voting network is secured against wiretapping. (A4)

Although multi-way elections and limited votes are preferable over simple binary votes in the long-term, the architecture of the designed system will focus on binary votes only. Although providing a cast-as-intended verifiability proof for a binary vote may not change heavily from one allowing multi-way elections, such a requirement may pose challenging requirements on the tallying process.

The submitted votes are of binary nature. (A5)

These five assumptions compose the cornerstones for the envisioned design of electronic voting on a blockchain. By making them explicit, reasoning on technical approaches becomes more concrete.

3.5 Public Bulletin Board

In electronic voting systems, a variety of data is usually made available on a public accessible storage: Not only public-private key-pairs used for encrypting votes but also information required for providing verifiability or proofs for the validity of the final tally. According to [43, 44], such a public bulletin board (PBB) provides the following properties:

1. It is an append-only data structure, i.e. information cannot be modified or altered.
2. It is public in the sense of being searchable by anyone.
3. It is consistent in its view for anyone accessing its information.

As a decentralised voting system is preferable in Switzerland as it is projecting its federalistic structure (cf. Section 3.3), a public distributed ledger providing a consistent view without having a single leader can act as a PBB. By using an appropriate implementation it is not only an append-only datastructure, searchable by anyone but also eventually consistent.

Considering Assumption A3, a permissioned setting can be used for achieving consensus, whereas each canton or municipality is running a single blockchain node. Implementations exist which provide private group-based (consortium) consensus, such as Ethereum with

its Turing-complete smart contract platform [11]. However, this smart contract platform poses a pair of limitation to perform cryptographic operations: (1) The available data types are only capable of storing a limited number of bytes and (2) are not yet supporting an implementation for big integers. This makes electronic voting using the ElGamal cryptosystem with finite cyclic groups infeasible as it relies on a large enough prime number as its modulus. In addition, computations on this platform can use at most a callstack depth of 1024[4]. For verifying proofs defined in cyclic fields, this might be too few. However, a custom blockchain can take into account these limitations and work around them from the very beginning.

3.6 Consensus

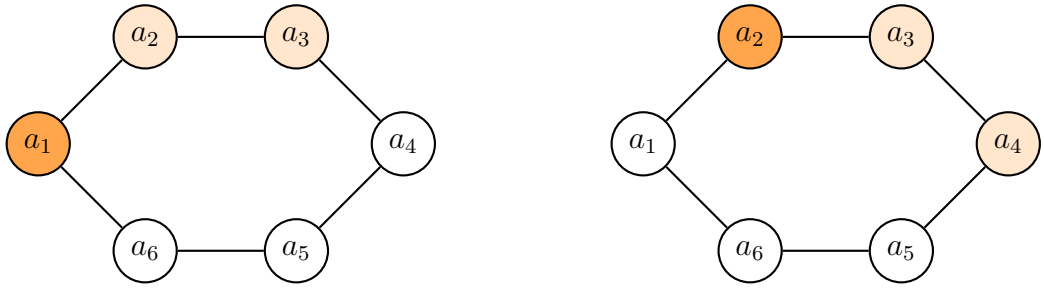
Finding an appropriate algorithm for reaching consensus in a permissioned blockchain setup is not straightforward. In PBFT, PoA and Tendermint, duplicated votes (cf. *A1*) must be detected on application level, allowing even invalid votes to be incorporated into the global state of the voting system. In contrast, Avalanche provides the notion of conflicting transactions and thus transitive conflict sets, which can be defined application-specific but operate directly on the protocol level. With respect to electronic voting with a single submission, transactions could contain the vote of a particular user $\mathcal{T}_{n,i}$ and a conflict set $\mathcal{P}_{\mathcal{T}}$ could be defined as containing all k transactions from the same user $\mathcal{T}_n$ which were submitted before or after the i-th one:

$$\mathcal{P}_{\mathcal{T}} = \{\mathcal{T}_{n,0}, \ldots, \mathcal{T}_{n,j}, \ldots \mathcal{T}_{n,k}\} \setminus \{\mathcal{T}_{n,i}\}, \text{ for } i = 0, 1, 2, \ldots, k \text{ and } j = 0, 1, 2, \ldots, k \text{ and } \forall i \neq j \tag{3.1}$$

The peer-to-peer network can then be utilised to build up confidence values for each transaction. The transaction with the highest confidence value of the conflict set would then be considered as the counting vote. Despite this preferable mechanism, considering Avalanche as consensus algorithm for a voting system comes with certain drawbacks: Avalanche itself does not directly impose the typical tree-like data structure for its data store, and thus, only a partial order is guaranteed between transactions [53].

As the Practical Byzantine Failure Tolerance algorithm replicates state machines, the electronic voting system should be built on top of such. However, state machines may not necessarily be required to implement electronic voting systems. Tendermint with its voting procedure for finding consensus on the next block is also quite complex. On the other hand, Ethereum's proof-of-authority algorithm Clique[5] provides a straight-forward way to implement consensus for blockchains due to its simplistic protocol and is thus a good starting point to reach consensus in a PoA setting. Although it originally comes with mechanisms to vote on the set of authority nodes allowed to sign blocks of transactions, this ability is not required in the electronic voting system as by Assumption A3 and can therefore be discarded.

[4] `https://solidity.readthedocs.io/en/v0.4.21/security-considerations.html`
[5] `https://github.com/ethereum/EIPs/issues/225`



(a) Epoch 1: a_1 is leader, a_2, a_3 are co-leaders (b) Epoch 2: a_2 is leader, a_3, a_4 are co-leaders

Figure 3.1: Clique's leader election with signer limit set to two

3.6.1 Leader and Co-Leader

In Clique, time is split up in periods of fixed length, defined as a system parameter. In each period, a well-defined set of nodes is authorised to sign blocks: A leader and a number of co-leaders. Each one is allowed to sign and broadcast blocks. Algorithm 1 specifies how a node recognises its leader role: By using the current block number and the total number of nodes, it calculates whether its index is currently seen as a leader. Algorithm 2 defines whether a node is allowed to be a co-leader of the current leader. Its index must be in the range of the leader index plus one and the maximum number of blocks any node can sign consecutively. In Figure 3.1, the block period is set to two, so that always two co-leaders exist in each epoch.

Algorithm 1: *isLeader*: Returns true, if the node is a leader for the current epoch. False otherwise.

Data: $SIGNER\_IDX$, the index of of the node in the list of authorised nodes; $SIGNER\_COUNT$, the total number of authorised nodes; $BLOCK\_NUMBER$, the number of the current block.

begin

  return $SIGNER\_IDX = BLOCK\_NUMBER \bmod SIGNER\_COUNT$

Algorithm 2: *isCoAuthority*: Returns true, if the node is a co-authority for the current epoch. False otherwise.

Data: $SIGNER\_IDX$, the index of of the node in the list of authorised nodes; $SIGNER\_COUNT$, the total number of authorised nodes; $SIGNER\_LIMIT$, the number of consecutive blocks out of which a signer may only sign one; $BLOCK\_NUMBER$, the number of the current block.

begin

  lowerBound $\leftarrow \left(BLOCK\_NUMBER \bmod SIGNER\_COUNT\right) + 1$

  upperBound $\leftarrow$ lowerBound $+ SIGNER\_LIMIT$

  return $SIGNER\_IDX \in \left[lowerBound, ..., upperBound\right]$

3.6.2 Transactions and Blocks

Whenever a node receives a new transaction, the node validates the transaction for correctness of its parameters. In addition, the node verifies that the transaction does not exist yet and avoids processing the transaction if it is already known in its local data store. Eventually, if the node is a leader or co-leader, it appends the transaction to the local store $\mathcal{T}$ containing all transactions which should be appended to a block once a period has ended. Then, the received transaction is broadcast to all other networks participants to ensure that (1) the leader or other co-leaders learn about the transaction, and (2) to avoid race conditions between blocks of a leader not containing the transaction and a block from the co-leader containing the transaction.

Algorithm 3: *onTransactionReceive*: Invoked whenever a transaction is received from another signer

```
Data: t_i: The transaction sent to the node, 𝒯 the set of transactions to form a
      block of
begin
    if !isValid(t_i) then
        return
    if isKnown(t_i) then
        return
    if isLeader() || isCoAuthority() then
        𝒯 ← 𝒯 ∪ t_i
    broadcast(t_i)
    return
```

A similar mechanism is applied when a block is received, broadcast from any of the authorised signers. This procedure is outlined in Algorithm 4: Each block is verified for its validity, i.e. whether it is signed appropriately and whether the references of the transactions as well as the block are correct. Eventually, the new block is appended to the canonical chain.

Algorithm 4: *onBlockReceive*: Invoked whenever a block is received by a node.

```
Data: ℬ_{S_i}, the block ℬ signed by signer S_i.
begin
    if !isValid(ℬ_{S_i}) then
        return
    append(ℬ_S)
    return
```

3.6.3 Block Signing

Signing a block is only attempted if an authorised node is currently either a leader or a co-leader. Then, the node waits until the period of the current epoch has ended and immediately builds the block with all known transactions. To reduce the number of forks

occurring when both the leader and its co-authorities announce a block at the same time, co-authorities are urged to delay their broadcast by a small amount of time, allowing the other nodes to receive the block from the leader first.

Algorithm 5: *sign*: The signing loop of any node

Data: t_i: The transaction sent to the node, $\mathcal{T}$ the set of transactions to append to a block

```
begin
    while true do
        isLeader ← isLeader()
        isCoAuthority ← isCoAuthority()
        if ! isLeader || ! isCoAuthority then
            continue
        nextRun ← parentBlock.timestamp + block.period
        if now < nextRun then
            continue
        if ! isLeader then
            sleep()
        B ← createBlock(T)
        B_S ← signBlock(B)
        append(B_S)
        broadcast(B_S)
```

3.6.4 Missing Transactions

Eventually, as a node can be out-of-sync with parts of the network, it may receive blocks referencing a parent which the node has not yet learned about. In such a case, it can query the node from which it has received the block, as the sender is guaranteed to know the parent as it has referenced it. This procedure can be repeated until the querying node finds a parent block referenced which it stores locally.

3.6.5 Forks and Fork Resolution

As multiple authorities can propose their blocks in a particular epoch, forks in the blockchain can occur [22]. In Ethereum, the Greedy Heaviest-Observed Sub-Tree (GHOST) protocol [56] is used to resolve such conflicts. In short, it chooses the heaviest subtree of each block for building the main chain, for a specific notion of weight. By assigning weights to each block such that the block of the leader weighs more than the ones of its co-authorities, the main chain can be constructed. By further taking into account the number of so-called uncle blocks, i.e. all other children of the grandparent of the block considered, blocks which have been referenced multiple times as a parent are preferred during traversal. Consider the chain shown in Figure 3.2 and suppose the GHOST protocol takes into account the number of direct uncles and children as its weight parameter.

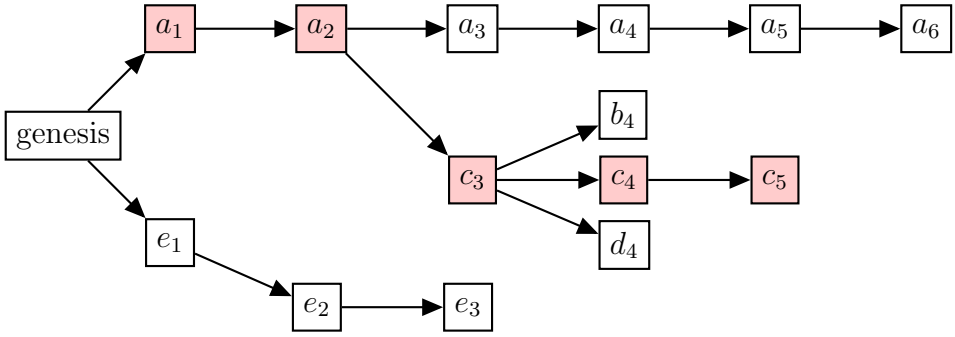


Figure 3.2: A chain of blocks with its heaviest chain marked in red

Starting from the blocks with the lowest height, node c_4 has a weight of 2 as it has one child and one uncle (a_3). Node c_3 has a weight of 3, as it has three children but no uncle. The weight for node a_3 is the number of its children, i.e. 1. Hence, the decision on the branch at a_2 is made in favour for c_3 as its weight (3) is heavier than the one from a_3 (1).

To further make preference for blocks which have been signed by a leader, Clique specifies a doubled weight for blocks being signed by a leader. In particular, Ethereum's Clique implementation is calculating up to seven levels[6] ahead for computing the number of uncles of a block.

3.7 Data Model

Although the broader concept of blocks and transactions are common in blockchains, their meaning differs highly depending on the use cases they support in their specific fields. Thus, a brief overview of the created data model is given in the following.

3.7.1 Transactions

A transaction is the actual payload of the blockchain. Besides a hash of the content identifying the transaction, it stores information of what kind of payload is embodied and the payload itself. Thus, the set (T_{Vo}, T_V, T_{Vc}) of transaction types defines what is intended by the payload it contains. The transaction specifying T_{Vo} indicates that all further received transactions should be considered as votes submitted after the election period has started. T_{Vc} defines a transaction which indicates that the voting period has ended and the election has been closed from accepting any further transactions. T_V is the actual voting transaction containing an actual vote from an end-user.

3.7.2 Blocks

A block is the body in which a set of transactions is contained. A block contains a timestamp at which it was created, the transactions as payload, and a hash referring

[6] `https://github.com/ethereum/wiki/wiki/Design-Rationale#uncle-incentivization`

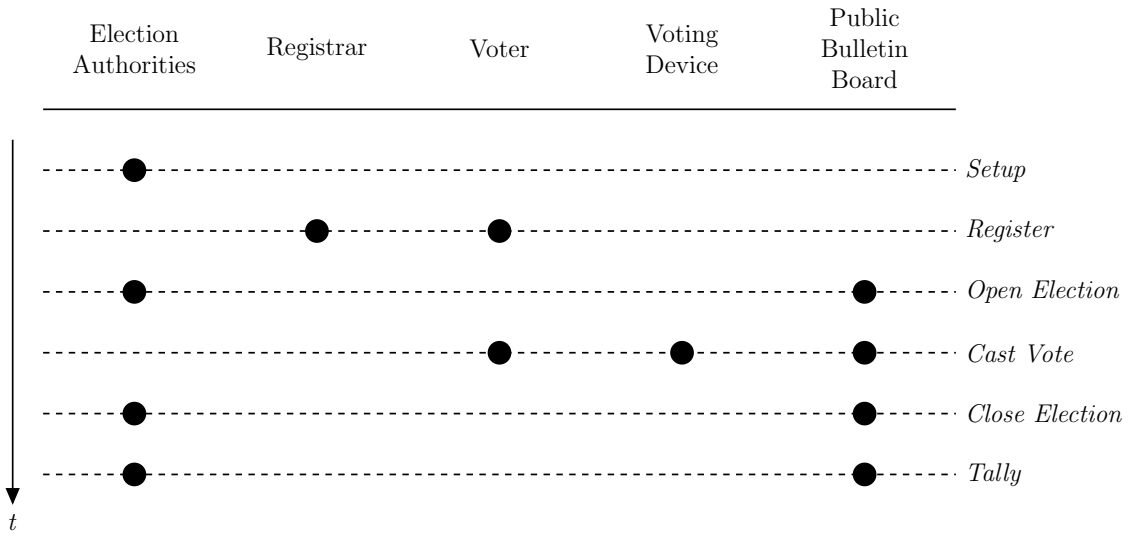


Figure 3.3: The parties involved in the election's different phases

to the identifier of its parent. Therefore, the chain of blocks can be built. The genesis block is a special block instantiation and specifies all system-relevant parameters for the blockchain. As such, it is used as root of the blockchain and the hash of its contents forms the first reference to which all direct child-blocks refer to. Any change applied to the configuration of the genesis block or to the content of all other blocks will lead to a different hash and therefore a different chain of blocks.

3.8 Voting Protocol

This thesis is, to our best knowledge, the first public implementation of a voting system supporting universal cast-as-intended verifiability on a blockchain. However, the core voting protocol is mostly adopted from [31] which defines the main protocol steps as outlined in this section. Figure 3.3 shows the five main parties involved in the voting protocol: (1) The election authorities, such as the participating municipalities or cantons, (2) the registrar, maintaining the voter registry and deciding about a voter's eligibility to take part in the election, (3) the voter who desires to submit his or her vote, (4) the voting device which is operated by the voter in order to cast a ballot, and (5) the bulletin board which is used as public accessible storage.

3.8.1 Setup of the Election

Protocol 1 - Setup: $\{\} \rightarrow \{SVI, \widetilde{SVI}, PVI, \sigma, (sk_e, pk_e)\}$.

The *Setup* protocol generates a public-private key-pair for the additive variant of the ElGamal cryptosystem. It consists of its private key sk_e and the corresponding public

key pk_e. In addition, it defines the space of secret universal cast-as-intended verification (UCIV) information (SVI), the corresponding space of voting-option dependent secret UCIV information $\widetilde{SVI}$, and the space of of public UCIV information PVI. In addition, the functions $\sigma_v : SVI \to \widetilde{SVI}$ mapping the secret UCIV information to a voting-option dependent secret UCIV information are specified [31].

3.8.2 Registration of Voters

Protocol 2 - Register: $\{vid, V, pk_e\} \to \{(uciv_s, uciv_p)^{vid}\}$.

The *Register* protocol generates the private and public UCIV information for a particular voter, based on its voter id vid, the set of voting options V, and the public election key pk_e. This protocol is invoked by a registrar who is acting independently from the voting authorities assigning the voter id vid to the voters. Consider a scenario in which the election authorities know which identity is identified by vid. In such a case, the authorities would be able to link the encrypted vote with the vid as the vid is submitted in the ballot. Since the authorities are also in possession of the election private key sk_e, the plaintext vote could be revealed and assigned to the identity masked by vid.

3.8.3 Casting a Vote

Protocol 3 - Cast: $\{vid, v, (\sigma_v(uciv_s), uciv_p)^{vid}, pk_e\} \to \{C^{vid}, M^{vid}, V^{vid}\}$.

As shown in Figure 3.3, an indicator is published on the bulletin board specifying that the voting application is accepting ballots prior to allow voters casting a vote. Then, for creating a ballot, the voting application requires the voter id vid, the selected voting option v, and the election public key pk_e. In addition, it receives as protocol input the output of the evaluation of the voting-option dependent function on the secret UCIV information $\sigma_v(uciv_s)$ as well as the corresponding public UCIV information $uciv_p$. With these arguments it generates the homomorphic additive ElGamal ciphertext C^{vid}. Correspondingly, it generates a universal cast-as-intended verification proof V^{vid} as described in more detail in [31]. A membership proof is guaranteeing that the encrypted vote either represents a zero or one, ensuring that only valid binary votes are submitted to the bulletin board. This property can be alternatively formulated as having a vote which is in the range $[0, 1]$. The proof itself is constructed as a Σ-protocol as shown in Figure 3.4. By using the technique outlined in Section 2.3.3, the interactive three-move proof is made non-interactive and added as M^{vid} along the cast-as-intended proof and the encrypted vote to the ballot. Then, the ballot is sent to the PBB.

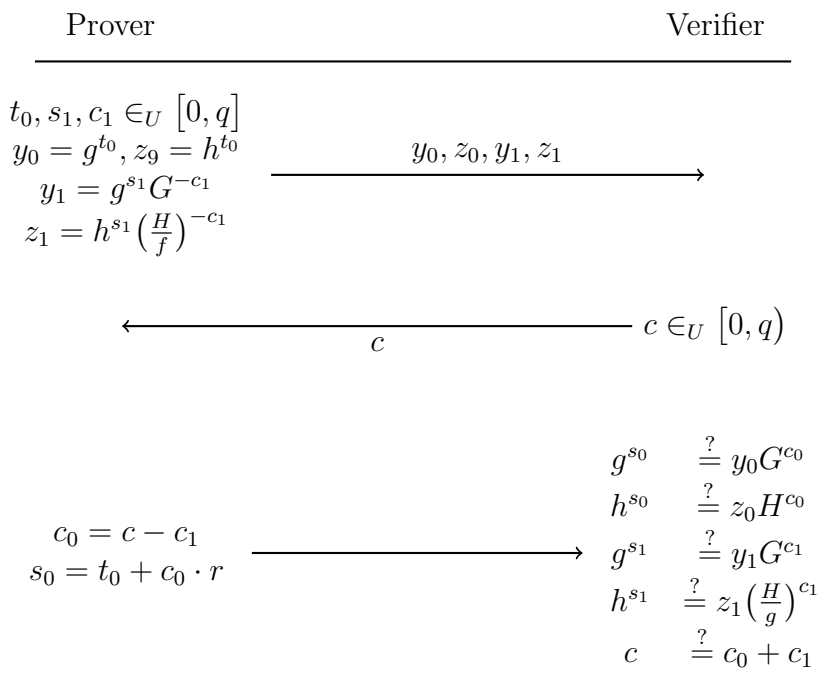


Figure 3.4: The Σ-protocol of the $[0, 1]$ range membership proof for an additive homomorphic ElGamal ciphertext $(G, H) = (g^r, h^r \cdot g^m)$ with r chosen randomly [45]

3.8.4 Calculating the Final Tally

Protocol 4 - Count: $\left\{\left(C^{vid}, M^{vid}, V^{vid}\right)^*, sk_e\right\} \rightarrow \left(T_s, T_o, T_i\right)$**.**

Before the voting authorities are calculating the final tally, the voting application must stop any further votes from being counted. Therefore, the indicator of having the vote opened must be terminated with a corresponding one specifying the election has been closed. Then, to calculate the final tally, the set of all triples $\left(C^{vid}, M^{vid}, V^{vid}\right)^*$ received on the PBB and the election private key sk_e are required as input arguments. Then, by homomorphically summing up all valid ciphertexts, the total number of voters having supported the voting question is obtained as T_s. By subtracting the total supporting votes from the total number of votes received, the opposing vote count is calculated T_o. As there might be ballots which should not be counted since the proofs contained in them are invalid, a total of invalid ballots is counted in T_i. Then, the triple $\left(T_s, T_o, T_i\right)$ is considered the final tally T.

Chapter 4

Implementation

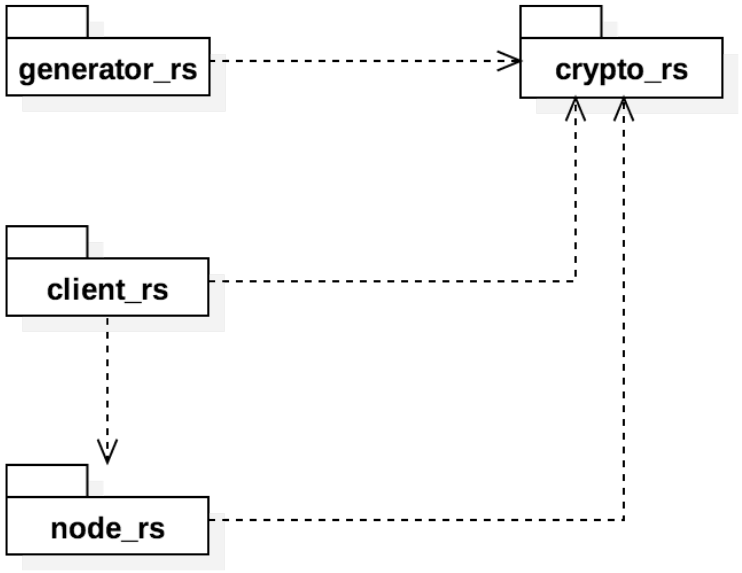


Figure 4.1: Diagram of all packages building up the electronic voting system

The instantiation of the previously described architecture is structured into four main packages as shown in Figure 4.1. The package `crypto_rs` provides the arithmetic foundation on which the additive homomorphic ElGamal cryptosystem is built. In addition, the implementations of non-interactive zero-knowledge proofs are contained. `generator_rs` creates all necessary cryptographic parameters, such as the private and public UCIV information and the election public-private key-pair. `node_rs` provides the PBB implemented as a blockchain whereas `client_rs` provides all functionality to administrate and participate in the election. All packages are implemented in `Rust`[1], "a systems programming language that ... prevents segfaults, and guarantees thread safety" [61].

[1] `https://www.rust-lang.org/en-US/`

4.1 crypto-rs

The `crypto_rs` package provides an implementation of modular arithmetic in cyclic groups, an ElGamal ciphertext and a corresponding key-pair, and the NIZKPK to verify the ciphertexts.

4.1.1 Modular Arithmetic

```
pub struct ModInt {
  /// The value.
  pub value: BigInt,
  /// The modulus.
  pub modulus: BigInt
}
```

Listing 4.1: Modular arithmetic data struture `ModInt`

In order to operate on a finite cyclic group, modular arithmetic is required. Since `Rust` does not come with a well supported library providing such functionality, an adoption of `AdderInteger`[2] is used. It is named `ModInteger` to represent its mathematical properties. Based on the `BigInt` implementation from `https://github.com/rust-num/num`, all basic arithmetic operations are redefined for modular arithmetic based on the data structure defined in Listing 4.1.

Inverse (a^{-1}): The inverse of $a^{-1} \mod c$ is only defined for values of a which are co-prime to c, i.e. have no common divisor. Then, the modular inverse can be found using the Extended Euclidean Algorithm for finding the greatest common divisor of a and c.

Negation ($-a$): $-a \mod c = (c - a) \mod c$.

Addition ($+$): $(a + b) \mod c = [(a \mod c) + (b \mod c)] \mod c$.

Subtraction ($-$): $(a - b) \mod c = [(a \mod c) - (b \mod c)] \mod c$.

Multiplication ($\cdot$): $(a \cdot b) \mod c = [(a \mod c) \cdot (b \mod c)] \mod c$.

Division ($/$) There is no division operation in modular arithmetic. However, there are modular inverses which can be used instead by exploiting the fact that $a \cdot \frac{1}{a} = 1$. Therefore, the division in modular arithmetic can be defined as $\frac{a \mod c}{b \mod c} = [(a \mod c) \cdot (b^{-1} \mod c)] \mod c$.

Exponentiation ($\hat{}$): $(a^b) \mod c = [(a \mod c)^b] \mod c$.

[2] `https://github.com/FreeAndFair/evoting-systems/tree/master/EVTs/adder`

4.1.2 ElGamal Homomorphic Encryption

Based on `ModInt`, the additive variant of the homomorphic ElGamal ciphertext is specified as a data structure containing only G, H and the random number r used during encryption. Then, the encryption, decryption, and homomorphic addition are implemented as specified in Definitions 3 and 4.

4.1.3 ElGamal Range Proof

In order to transform the ElGamal range proof (see Figure 3.4) to its non-interactive form, the requests and responses between the verifier and the prover must be abstracted. This abstraction is achieved by hashing the commitment and the corresponding value. The triple consisting of the commitment, the value, and the obtained hash are then the static proof for the value for which the proof was made. As there are numerous values that could be considered valid in a range proof, multiple applications of this procedure must be executed as shown in Algorithm 6. However, not only the generation of the proof requires multiple iterations, but also the verification procedure, as depicted in Algorithm 7.

4.1.4 Cast-as-Intended Verification Proof

Similarly to the range proof, the cast-as-intended verification proof is made non-interactive as well. Here as well, multiple iterations must be invoked, one for each available voting-dependent secret UCIV information ($uciv_s$). This is outlined in Algorithms 8 and 9.

4.2 generator-rs

The `generator_rs` binary is able to generate all required cryptographic material, such as the private and public UCIV information and the election key-pair. However, creating a new ElGamal key-pair is not yet cryptographically safe: As of today, there is no safe prime generator available for the used `BigInt` abstraction. In other words, the prime modulus p, its co-prime q as well as the private key x are currently hard-coded. Thus, `generator_rs` is a best-effort solution for being able to process the entire voting procedure. However, this must be replaced once an implementation is available.

In addition to generating an election key-pair, this package is further able to generate a set of public and secret UCIV information $(uciv_s, uciv_p)^*$. For ease of this proof-of-concept implementation, the voting option dependent function σ_v is already applied to the set of secret UCIV information $uciv_s$. Since computing the logarithm of a particular number is considered computationally expensive in the ElGamal cryptosystem, σ_v is the exponentiation function $F(x) = g^x$ as proposed in [31].

Algorithm 6: Creation of a non-interactive zero-knowledge proof of knowledge for proving that an ElGamal ciphertext is within the domain D.

Data: ElGamal public key parameters p, g, h, Plaintext m, Ciphertext $E = (G, H)$, Domain D of valid messages $\{0, 1\}$.

begin

- $sb \leftarrow g \| h \| G \| H$
- $t \in_{random} \mathbb{Z}_{q-1}$
- **for** $i \in |D|$ **do**
  - **if** $D_i = m$ **then**
    - $s_i \leftarrow 0$
    - $c_i \leftarrow 0$
    - $y_i = g^t$
    - $z_i = h^t$
  - **else**
    - $s_i \in_{random} \mathbb{Z}_{q-1}$
    - $c_i \in_{random} \mathbb{Z}_{q-1}$
    - $y_i \leftarrow g^{s_i} \cdot G^{-c_i}$
    - $z_i \leftarrow h^{s_i} \cdot \left(\frac{H}{g}\right)^{-c_i}$
- c $\leftarrow Hash(sb, y_1, z_1, y_2, z_2)$
- $c_0 \leftarrow c - c_1 - c_2$
- $s_{D_m} \leftarrow c_0 \cdot r + t$
- $c_{D_m} \leftarrow c_0$

return $s_1, s_2,\ c_1, c_2$

Algorithm 7: Verification of a non-interactive zero-knowledge proof of knowledge ensuring that the encrypted ciphertext value is within a given domain D.

Data: ElGamal public key parameters p, g, h, Ciphertext $E = (G, H)$, Domain D of valid messages $\{0, 1\}$, proof parameters $s_1, s_2,\ c_1, c_2$.

Result: True, on a valid proof for the specified ciphertext. False otherwise.

begin

- $sb \leftarrow g \| h \| G \| H$
- **for** $i \in |D|$ **do**
  - $y_i \leftarrow g^{s_i} \cdot G^{-c_i}$
  - $z_i \leftarrow h^{s_i} \cdot \left(\frac{H}{g}\right)^{-c_i}$
- c $\leftarrow Hash(sb, c_1, c_2, y_1, z_1, y_2, z_2)$

return $c_1 + c_2 \equiv c$

Algorithm 8: Creation of a non-interactive zero-knowledge proof of knowledge for proving that an ElGamal ciphertext represents the intended vote.

Data: ElGamal public key parameters p, g, h, r, pre-image set $uciv_s$, the corresponding image set $uciv_p$, Ciphertext $E = (G, H)$, the index of the chosen vote in the image set idx and the set of available voting options V.

begin

- $sb \leftarrow G \| H$
- **for** $i \in |uciv_s|$ **do**
  - **if** $V_i! = idx$ **then**
    - $s_1^i \in_{random} \mathbb{Z}_{q-1}$
    - $h_1^i \in_{random} \mathbb{Z}_{q-1}$
    - $r^i \in_{random} \mathbb{Z}_{q-1}$
    - $c_1^i \leftarrow g^{s_1^i} \cdot G^{-h_1^i}$
    - $c_2^i \leftarrow h^{s_1^i} \cdot \left(\frac{H}{g^{V_i}}\right)^{-h_1^i}$
  - **else**
    - $s_2^i \in_{random} \mathbb{Z}_{q-1}$
    - $h_2^i \in_{random} \mathbb{Z}_{q-1}$
    - $b^i \in_{random} \mathbb{Z}_{q-1}$
    - $c_1^i \leftarrow g^{b^i}$
    - $c_2^i \leftarrow h^{b^i}$
    - $r^i \leftarrow g^{s_2^i} \cdot uciv_p^i$
  - $sb \leftarrow sb \| c_1^i \| c_2^i \| r^i$
- c $\leftarrow Hash(sb)$
- **for** $i \in |uciv_s|$ **do**
  - **if** $V_i! = idx$ **then**
    - $h_2^i \leftarrow c - h_1^i$
    - $s_2^i \leftarrow r^i + (uciv_p^i)^{-h_2^i}$
  - **else**
    - $h_1^i \leftarrow h_2^i$
    - $s_1^i \leftarrow r^i + (r \cdot h_2^i)$

return $s_1, s_2,\ h_1, h_2, c$

Algorithm 9: Verification of a non-interactive zero-knowledge proof of knowledge ensuring that the encrypted ciphertext represents the intended vote.

Data: ElGamal public key parameters p, g, h, r, the image set $uciv_p$, Ciphertext $E = (G, H)$, and the set of available voting options V.

Result: True, on a valid proof for the specified ciphertext. False otherwise.

begin

- $sb \leftarrow G \| H$
- **for** $i \in |uciv_p|$ **do**
  - $c_1^i \leftarrow g^{s_1^i} \cdot G^{-h_1^i}$
  - $c_2^i \leftarrow h^{s_1^i} \cdot (\frac{H}{g^{V_i}})^{-h_1^i}$
  - $r^i \leftarrow g^{s_2^i} \cdot (uciv_p^i)^{-h_2^i}$
  - $sb \leftarrow sb \| c_1^i \| c_2^i \| r^i$
- $c\prime \leftarrow Hash(sb)$

return $c\prime \equiv c$

4.3 node-rs

The implementation of the blockchain acting as PBB is provided in `node_rs`. Thus, it requires `crypto_rs` as dependency, providing it with the data structures on which transactions and blocks are built upon. Figure 4.2 shows its components: ❶ Blockchain nodes communicate using the `Node RPC interface`. ❷ Votes are submitted to a dedicated `Client RPC interface`. Besides two threads listening for incoming connections, a thread pool ❸ also runs another thread signing blocks if the node is a leader or co-leader. ❹ The `Clique Protocol Handler` operates on the Proof-of-Authority level where it handles incoming messages and corresponding responses, creates transactions and blocks, and decides whether the node is a leader or co-leader for the current epoch. In addition, it holds a transaction buffer ❺ and its own instance of the actual blockchain ❻. It is worth noting that all three threads share the same instance of the protocol handler using a mutex in order to avoid race conditions amongst them. Therefore, all threads create decisions based on the same chain instance. This is also illustrated by the single interface to the `Clique Protocol Handler` in Figure 4.2.

4.3.1 Configuration

The `configuration` module contains utilities to read all required configuration into a representation which can be used as genesis block for the blockchain. It contains the following fields:

- **Version**: A version indicator following the constraints of semantic versioning.[3]
- **CliqueConfig**: The clique configuration contains system parameters which directly affect how the Clique protocol is applied. These are `blockPeriod` which defines in seconds how long an epoch lasts and `signerLimit` the number of blocks each node in the network is allowed to sign consecutively.
- **Sealer**: This is the set of listening addresses of all nodes participating in the network. Based on this information, each node will be able to determine his signer index and thus, whether it is a leader or co-leader for the current epoch.
- **PublicKey**: This is the election ElGamal public key pk_e used to encrypt all votes and verify the associated proofs.
- **PublicUciv**: This contains a vector of all public UCIV information $uciv_p$ which is required to verify the cast-as-intended verification proof of transactions.

During instantiation of the Clique protocol, the genesis block is hashed and the resulting value used to determine whether a node in the network is based on the same configuration. If a configuration value is different, also the hash will become different and thus, the nodes will never agree on the same canonical chain. Therefore, nodes with a different genesis block hash are excluded from communication.

[3] `http://semver.org`

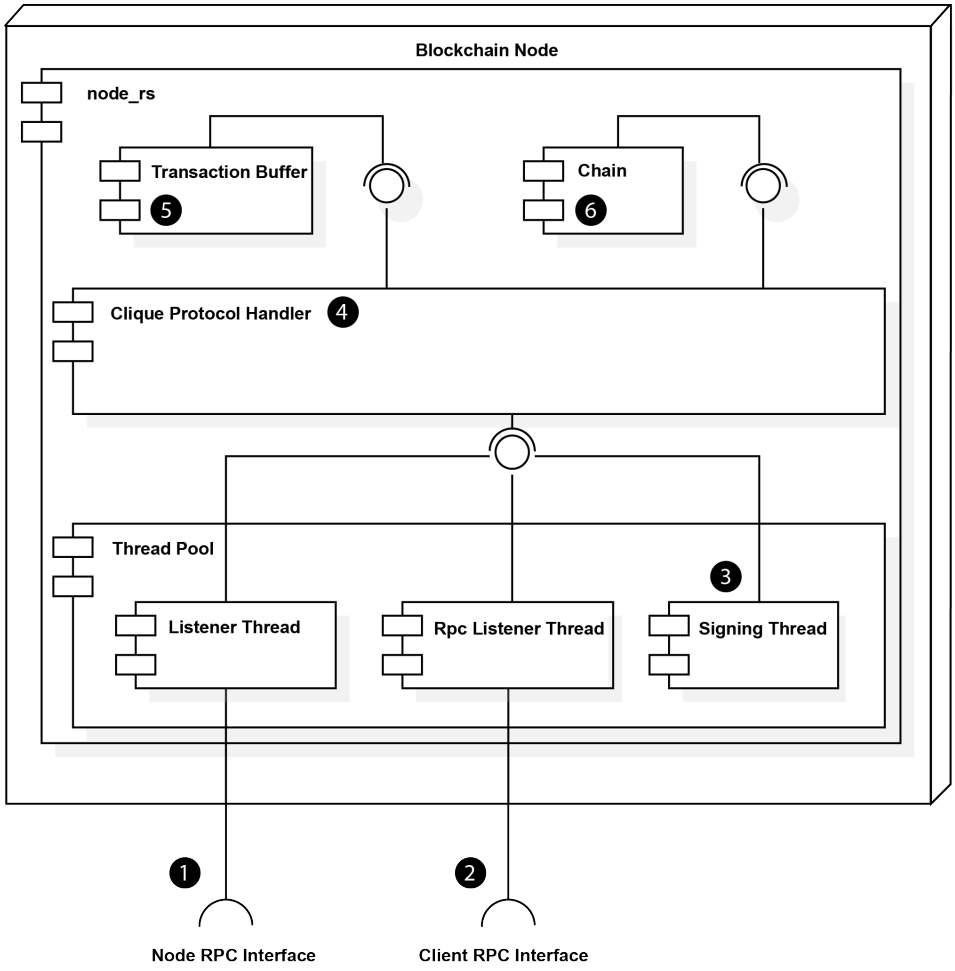


Figure 4.2: A blockchain node's architecture. ❶ ❷ are the external communication interfaces. ❸ is a thread signing blocks. ❹ is the actual proof-of-authority protocol handler with its transaction buffer ❺ and the blockchain ❻.

4.3.2 Chain

The data of the blockchain itself is stored on the heap. Thus, cycling references easily become a source of memory leaks[4]. Instead of using a tree-based data structure to build up the chain, an adjacent matrix is created, containing on the y-axis all block identifiers and on the x-axis the corresponding children identifiers of the block. In addition, its underlying data structure of a `HashMap` ensures $\mathcal{O}(1)$[5] cost when reading for a particular key. Therefore, traversing the chain is not a complex task: In order to find the last block of the canonical chain, the `HeaviestBlockWalker` attempts to traverse the chain from the genesis block onward to the block with the biggest height, i.e. the deepest level in the chain. Only this block is then visited using the `HeaviestBlockVisitor`. To count the votes stored on the chain, a `LongestPathWalker` is implemented as follows: First, it finds the last block of the canonical chain, starting from the Genesis block. Once found, it traverses each block bottom-up back to the Genesis block. Each block is then visited using

[4] `https://doc.rust-lang.org/book/second-edition/ch15-06-reference-cycles.html`
[5] `https://doc.rust-lang.org/1.22.1/std/collections/index.html#maps`

the `SumCipherTextVisitor` which homomorphically sums up all votes on the canonical chain. These walkers differentiate mainly in how their visitors are invoked: Whereas the `HeaviestBlockWalker` only invokes the visitor with the heaviest block found, the `LongestPathWalker` visits all blocks along the canonical chain.

4.3.3 Peer-to-Peer Networking

The `p2p` module includes the foundation of `node_rs`: It defines TCP streams as connection channels between nodes, a thread pool in which multiple tasks can be handled concurrently, and a codec transforming incoming and outgoing messages into their appropriate formats.

As shown in Figure 4.2, each `node_rs` instance owns three threads in which the main operations are applied to a shared instance of the blockchain. In Rust, a TCP stream cannot be split up into two distinct parts for reading and writing[6]. Thus, sending and receiving data on the same stream is hardly achievable. To overcome this implementation-wise drawback, outgoing traffic is sent through a dedicated TCP stream instance and therefore has a different IP-address and port. However, as incoming connections are established from other nodes, it is possible to send responses on the same stream by first closing the reading half and then flush bytes to the writing half. Due to this inconsistent behaviour, the `Listener Thread` would be able to only successfully authenticate connections on streams it initially created, as it connects to the IP addresses defined in the Genesis configuration. Thus, incoming connections not previously specified in the configuration will be terminated. Based on this, the `Listener Thread` currently avoids authenticating and authorising incoming connections from other signing nodes completely.

The format used to exchange information between nodes is JSON. Thus, all messages are encoded to JSON by using an appropriate codec before they are serialised into a byte sequence and sent over the underlying TCP streams. Decoding is applied vice versa, where it is ensured that only valid messages are passed further to the protocol handler. As this is defined in the networking package, the actual serialisation format can be adjusted without affecting the other implementation of `node_rs`.

4.3.4 Protocol

Communication between nodes follows a strict protocol defined in the `Clique Protocol Handler`. As clients submitting votes expect different responses than nodes in the blockchain network, they interact with their dedicated interfaces, handled by the `Listener Thread` and `Rpc Listener Thread`, respectively. Transactions received by a node in the network are always broadcast to other nodes, regardless of their actual payload. Therefore, not only epoch leaders but also their co-leaders will be notified about a new transaction.

[6] `https://github.com/rust-lang/rust/issues/11165`

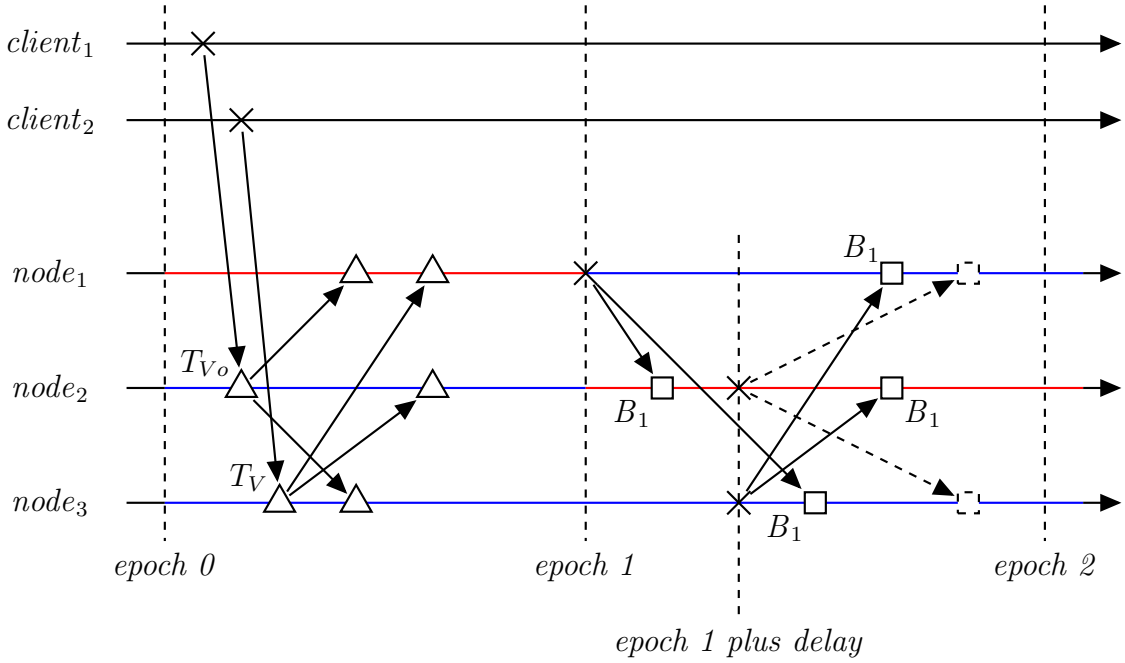


Figure 4.3: Distribution of messages submitted by clients and handled by nodes according to the Clique protocol. Nodes having lines depicted in red are epoch leaders, lines in blue represent their co-leaders

As described in Section 3.8, an election must be opened before any voting transaction can be accepted by the network. The corresponding communication sequence is illustrated in Figure 4.3. Consider the $client_1$ to be run by the election authorities. $Client_2$, however, is operated by an eligible voter. Further assume that the election setup procedure has been completed. Then, the communication procedure is as follows: The election authorities open the vote by sending an open vote transaction T_{Vo}. Subsequently, a voter submits his decision by sending T_V. Both transactions are immediately broadcast to all other nodes in the network, i.e. $node_1$ and $node_3$, and $node_1$ and $node_2$, respectively. All nodes then add the received transactions to their internal `Transaction Buffer` since they act as either an epoch leader or co-leader. As soon as *epoch 1* has arrived, $node_1$ signs a block with all transactions (i.e. T_{Vo} and T_V) and broadcasts them to its co-leaders. $Node_2$ and $node_3$, however, wait a defined delay before also signing and broadcasting their blocks. Note, that $node_2$ has received the signed block from $node_1$ before the delay is over. Therefore, it will avoid broadcasting its own block. $Node_3$, however, has not yet received any signed block instance from the other nodes once the delay is over, and will therefore start to broadcast his block. Since both, $node_1$ and $node_2$, have received the same block ahead of receiving the copy from $node_3$, they will not add the copy to their chain anymore.

Besides announcing blocks and transactions to nodes, the `Clique Protocol Handler` is further able to send an entire copy of its own chain to other nodes. During start up, each node will broadcast a `ChainRequest` to its other peers defined in the Genesis configuration. On reception, a node will return an entire copy of its chain to the requester. Then, a node will replace its own chain with the received one, *iff* its genesis hash equals to the own one, and the depth of the canonical chain is longer than the own.

4.4 client-rs

In the current prototype, all functionality for administrating elections, submitting votes, and obtaining a final tally is combined into one client application.

4.4.1 Vote Administration

As outlined in Section 3.8, election authorities need to explicitly send an indicator to open the ballot box represented by the PBB. By sending an `OpenVote` message to the bulletin board, the election is officially opened and incoming vote transactions will be counted towards the final tally. Transactions received beforehand will not be counted in the final result. Once the election period is over, the authorities need to send a `CloseVote` transaction to the bulletin board. Similarly, vote transactions submitted after the `CloseVote` transaction are not counted towards the final tally.

4.4.2 Vote Selection and Submission

Submitting a vote is not a complex task. Voters only need to be in possession of the election public key pk_e and their associated private and public UCIV information pair $(uciv_s, uciv_p)^{vid}$. Then, they can type in either `yes` or `no` as answer to the voting question. This value is then transformed in its binary representation 1 or 0, respectively. This binary vote is further encrypted on the voter's client device, and the range proof (cf. Algorithm 6) as well as the cast-as-intended verification proof (cf. Algorithm 8) are created. Before submitting the vote to the blockchain, both proofs are validated. Only if this validation was successful, the vote will be submitted to a node of the voting network.

4.4.3 Obtaining a Final Tally

Once the election period is over and the authorities have submitted a `CloseVote` transaction, the next step is to compute the final tally of the election. By sending a `RequestTally` to a blockchain node, a traversal from the root to the end of its current canonical chain is initiated. By using the `SumCipherTextVisitor`, each block's vote transactions T_V are summed up using the operations outlined in Definition 4. Beforehand, each vote transaction is validated by their associated proofs. In case a proof fails to evaluate successfully, the transaction is counted towards invalid votes. Only transactions in between an `OpenVote` and `CloseVote` transaction are counted. Once the entire canonical chain has been traversed, a response containing the amount of successful, invalid and total votes is returned. If no `CloseVote` transaction is observed during the traversal of the canonical chain, zero-values are returned for the above metrics.

Chapter 5

Evaluation

Switzerland specifies a substantial list of requirements which need to be fulfilled by any voting system targeting nationwide elections [25].This chapter will evaluate whether the stated subset of these requirements from Section 3.2 have been fulfilled and the assumptions from Section 3.4 still hold. Additionally, an outline of the runtime and storage complexity is provided.

5.1 Requirements

Requirement R1 The main goal of this thesis was to achieve *cast-as-intended* verifiability, allowing voters to verify that their encrypted votes contain the selection they have made. By creating a non-interactive zero-knowledge proof of knowledge for this property and verifying it before submitting the vote to the PBB, a voter can be ensured that the vote is appropriately encrypted. Considering a scenario where an adversary has access to the binary distributed to the voter and has adjusted it to always show a successful cast-as-intended verification proof to the voter regardless of his choice. An attentive voter would also be able to detect such a case: As of today, it is usual that software vendors provide signed binaries and corresponding checksums which can be used to verify the integrity of the executed binary.

Recorded-as-cast verifiability can be ensured as well: If a voter obtains the identifier of the transaction submitted to the blockchain, it can be queried after submission. As such, the voter can verify the associated proofs for correctness. However, as the transactions are not signed with a voter-dependent value the integrity verification of the vote on the blockchain becomes poor. However, the transaction's integrity can be verified by computing the hash of the transaction before submitting it to the blockchain and then asserting for equality during verification.

Counted-as-recorded is not yet provided by the implementation proposed as this thesis primarily focused on providing *cast-as-intended* verifiability. However, the protocol outlined in Section 3.8 can be extended in the last step by computing a non-interactive zero-knowledge proof of knowledge for the correct tabulation and publishing it to the bulletin board as well.

Individual verifiability, can be provided if the voter has a means of retrieving his vote from the blockchain and evaluating it for correctness using the associated proofs. Universal verifiability may be provided if a tabulation proof of the final tally is published to the PBB. Due to these limitations, the proposed implementation does not yet fulfil end-to-end verifiability completely.

Requirement R2 Although votes are locally encrypted on the voters' end-user devices, election authorities are still able to decrypt individual votes if they have a means of querying the blockchain for individual transactions. As this is suggested for providing *recorded-as-cast* verifiability, it will be hard to refuse this access to election authorities, especially as they provide the infrastructure running the blockchain. One can overcome this limitation by establishing multiparty computation as suggested in the work of [31]. In addition, [31] states that the suggested voting protocol on which this work is based, trusts the election authorities for privacy. Although the election authorities may decrypt any vote without using multiparty computation, confidentiality of the vote is not broken as the voter id *vid* is only linked to an identity by the registrar. As long as the registrar is independent from the election authorities and avoids collaborating with a malicious authority, ballot secrecy is ensured.

Requirement R3 In terms of the steps a voter has to perform to participate in the decision process of an election, the proposed system is similar to the current paper-based postal voting. Once the set of public and private UCIV information $(uciv_s, uciv_p)^{vid}$ and the election public key pk_e is obtained, only a single, self-contained step needs to be performed to submit a vote. With the introduction of e-identity solutions[1] the initial complexity of the registration step for a voter can be reduced. It is possible that the UCIV information $(uciv_s, uciv_p)^{vid}$ is obtained over electronic channels from an independent set of registrars, making a printer facility for paper-based voting cards obsolete.

Requirement R4 As the blockchain is currently not implemented as an account-based model, voters do not need to pre-fund an account in order to vote. Although this fulfils R4, it also allows for submitting arbitrary transactions to the blockchain. This drawback is further discussed in Section 5.4.

5.2 Assumptions

Most of the assumptions specified in Section 3.4 still hold with the current implementation. However, not all of them are enforced: In practice, voters can submit arbitrary votes with their voter identity *vid* to the blockchain, contrasting to Assumption A1. Both Assumptions A2 and A3 are enforced: Whereas the first is fulfilled by assigning each voter a voter identity *vid*, the latter is expressed in the configuration parameter `Sealer` required in the Genesis configuration of each node instance. However, the current proof-of-concept implementation does not provide encrypted communication between nodes as assumed in

[1] `https://www.bj.admin.ch/bj/de/home/staat/gesetzgebung/e-id.html`

Assumption A4 as this does not affect the procedure of the core voting protocol. However, a non-secured channel would allow an adversary to link the encrypted vote with the IP address of its origin, and thus, the voter. Considering further, that an adversary could be in possession of the election private key, the plaintext vote could be restored. With providing an end-user client application allowing to submit binary votes only, Assumption A5 is fulfilled.

5.3 Runtime and Storage Complexity

When performing real-world elections, it is essential that generating the required cryptographic material is performed within a reasonable time. Optimally, the runtime is at worst linear in the number of voters, i.e. $\mathcal{O}(n)$ for n voters. Obtaining time metrics in Rust is a built-in feature and provided by its benchmarking suite[2].

5.3.1 UCIV Information Generation Complexity

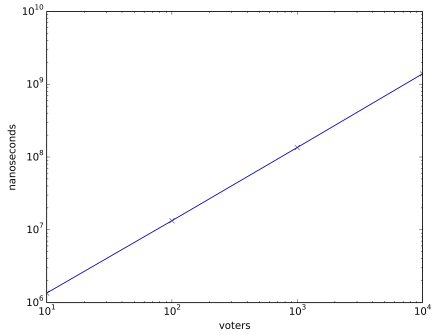
Figure 5.1 shows the runtime for generating the set of private and public UCIV information $(uciv_s, uciv_p)$ for different amounts of voters and voting options. In particular, four different amounts of voters have been evaluated: 10, 100, 1000, and 10000. First, as depicted in all three Figures 5.1a, 5.1b, and 5.1c, the runtime complexity is linear to the number of voters. Second, the time required to generate the UCIV information behaves linear in the number of voting options. Figure 5.1d shows the approximately linear behaviour of the runtime when generating the UCIV information for multiple amounts of voters. Based on these numbers, creating a linear approximation for generating UCIV information for the average Swiss electorate of 5357836 people in 2017[3] yields an approximate of 743 seconds ($\sim$ 12 minutes) for two voting options, 1069 seconds ($\sim$ 18 minutes) for three voting options, and 1464 seconds ($\sim$ 24 minutes) for four voting options.

5.3.2 Storage Complexity

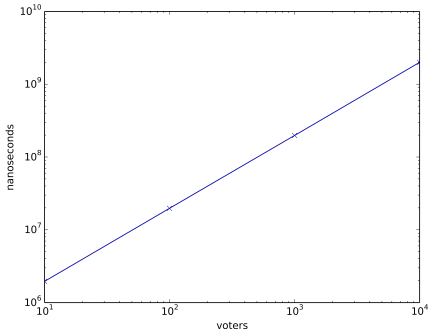
In addition to the runtime required for generating the UCIV information, the storage complexity of the blockchain with respect to the number of voters may yield specific hardware requirements. Therefore, Figure 5.2a shows the number of KBytes required to store 10, 100, 1000, and 10000 vote transactions T_V with their corresponding proofs. Unsurprisingly, the storage complexity is also linear to the number of transactions. Figure 5.2b shows the time in seconds required to handle these votes by the Clique protocol. Instead of being $\mathcal{O}(1)$ as one might expect, it is also linear in the number of transactions as the `Clique Protocol Handler` verifies whether a transaction is already known and thus needs to query the set of known transactions.

[2]See Appendix C for more information on how the benchmark metrics were created.

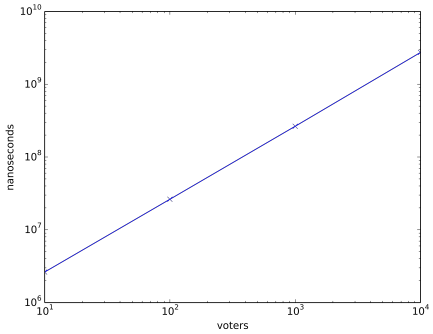
[3]`https://www.bfs.admin.ch/bfs/de/home/statistiken/politik/abstimmungen/stimmbeteiligung.html`



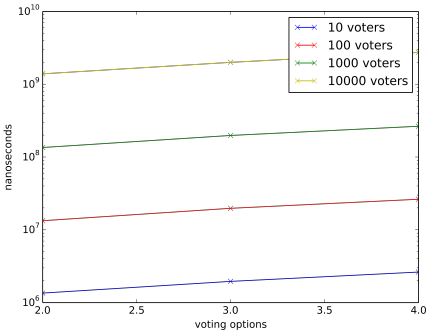
(a) two voting options



(b) three voting options



(c) four voting options



(d) 10, 100, 1000, and 10000 voters

Figure 5.1: Runtime for generating public and private UCIV information $(uciv_s, uciv_p)^*$ for different numbers of voters and voting options

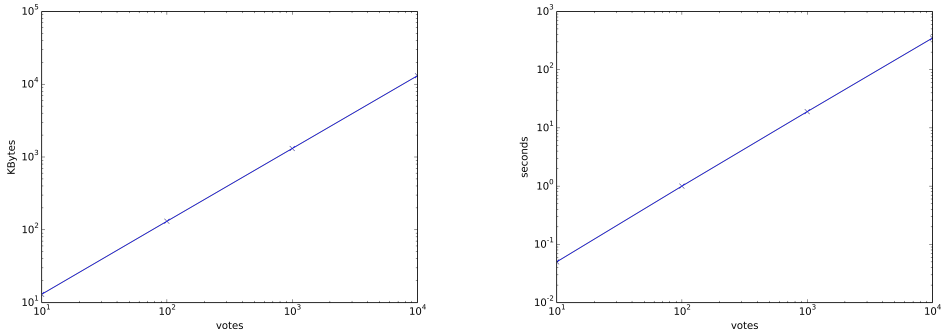
5.3.3 Runtime for Obtaining the Final Result

Further, it is crucial that obtaining the final tally is achievable in a timely manner. Considering a blockchain with 10, 90, 900, and 9000 transactions in consecutive blocks, the result of an election was retrieved in less than one second.

5.4 Security Considerations

Section 3.1 of this work has elaborated on some central points which may affect the security of the voting system's implementation. Distributing the voting application in form of a signed binary brings all the advantages of an established operating system's integrity protection: The binary's signature can be verified prior to its execution without needing the voter to perform a series of complex tasks. However, as opposed to a web-application, developers may need to be registered to a vendor specific developer program before being able to distribute the voting application through a company-controlled app store[4]. In addition, binaries need to be compiled and designed for specific architectures and devices, making it harder to maintain. In contrast, the browser's runtime environment is hardly to be trusted for privacy critical applications due to installed extensions which may alter

[4]Such as `https://developer.apple.com/support/code-signing/`



(a) Memory Usage in KBytes (b) Runtime in seconds

Figure 5.2: Memory usage and corresponding runtime to add vote transactions

its behaviour. Thus, a risk analysis weighing one approach against the other may support a final decision for the runtime platform of a production-ready implementation of the presented voting protocol.

As a PoA algorithm, Clique can tolerate up to $\frac{N}{2} - 1$ malicious participants [22]. Thus, the voting system can reach consensus even with dishonest election authorities. Since voters can query the PBB for their submitted vote, a malicious authority, censoring arbitrary transactions by abstaining from broadcasting them to other peers, can be identified. In such a case, voters can resubmit their vote to another authority and query the PBB for their vote, being assured that the vote was registered.

A non-negligible limitation in this work is currently caused due to the homomorphic addition of the ciphertexts on the blockchain: As of Definition 4, the randomness used in each ciphertext must be provided for correct homomorphic tabulation. As this calculation is executed on a node running an instance of the PBB, the randomness is submitted along the encrypted vote and the corresponding proofs to the blockchain. However, by making this information public, an adversary obtains confidential information which can cause a significant shrinkage of the solution space it has to search through to decrypt a ciphertext.

Sybil attacks aim at gaining an advantage over honest players by forging multiple identities [29]. As electronic voting systems tabulate supporting and opposing votes, it is crucial that such attacks can be prevented or at least detected. In the proposed voting protocol, mounting such an attack is difficult: If the registrar distributing the required UCIV information $(uciv_s, uciv_p)^{vid}$ is honest and able to detect forged identities, then an adversary will need to generate this information pair by itself as the registrar will abstain from providing it. Although a valid set of proofs can be generated by the adversary, verifying will still fail for the cast-as-intended verification proof: Assume the adversary obtains a voter id *vid* known by the electronic voting system and submits a correctly encrypted vote as well as the set of proofs to the blockchain. As the network nodes require the set of public UCIV information for each voter prior to the election in order to validate incoming votes, the generated proof of the adversary will not match the original parameters, making the proof evaluate to false with a high probability.

Replay Attacks [60] aim at reusing sent protocol messages to achieve a malicious system behaviour without the adversary being able to produce or read them. The proposed voting protocol is not affected by replay attacks as each voter is only allowed to vote once as per Assumption A1: A replayed message is not accepted by the blockchain nodes as any subsequent vote transactions T_V from the same voter are discarded.

Chapter 6

Summary and Conclusion

This thesis focused on bringing cast-as-intended verifiability to blockchain-based electronic voting. Prior work in this research area has shown that often more than one interaction is required to verify that a particular encrypted ballot is representing the intended voting decision. However, such solutions are hardly applicable to peer-to-peer networks, the underlying network topology of distributed ledgers. Therefore, an non-interactive approach is required. It was shown that by storing the non-interactive zero-knowledge proof of knowledge proposed by [31] on the blockchain, cast-as-intended verifiability is guaranteed while still maintaining a limited notion of privacy. In addition, tabulation of the final tally is performed directly on the blockchain, distributing the trust for its integrity among multiple authorities. However, the opposing properties of verifiability and privacy also revealed challenges for providing recorded-as-cast and counted-as-recorded verifiability in a distributed setup: Ballot privacy can be reduced by homomorphically tabulating directly on the blockchain. With regards to Swiss National Votes, extensions to the current proof-of-concept implementation must be provided in order to fulfil the fundamental property of end-to-end verifiability.

6.1 Future Work

Although this work presents a working solution for an electronic voting system which provides cast-as-intended verifiability, some implementation-specific simplifications require attention prior to performing a nationwide election.

6.1.1 Verifiability and Ballot Secrecy

As discussed in Section 5.1, counted-as-recorded verifiability would require a tabulation proof for the final tally to ensure end-to-end verifiability. Similarly to the currently employed NIKZPKs, a commitment to the obtained plaintext sum with respect to its corresponding ciphertext could be generated and published to the blockchain, being verifiable for any auditing entity. Further, performing the homomorphic addition on the blockchain

leaks information about the used randomness of the ciphertext in a public manner, reducing the solution space an adversary needs to consider when trying to decrypt a vote. Encrypting this information with an additional key only known to the election authorities could solve this issue: As the randomness is not required to validate the proofs associated to the encrypted vote, cast-as-intended verifiability is guaranteed without reducing the privacy of any voter. However, in the current implementation, any voting authority is in possession of the election private key sk_e. Thus, decrypting an arbitrary vote is possible. By establishing multiparty computation, this capability is removed: The ElGamal cryptosystem is adjustable to generate composite private keys requiring a threshold of k out of n authorities in order to decrypt any given value.

6.1.2 Authentication and Authorisation

The current prototype lacks authentication of incoming connections due to the limitations described in Section 4.3.3. Hence, authorisation is omitted as well. In a production ready application, messages could be signed with an asymmetric key-pair by each election authority. Thus, illegitimate requests attempted by non-authority entities could be filtered out.

6.1.3 Authority Coordination

In the current architecture, it is sufficient that a single election authority is submitting a open vote transaction T_{Vo} to the blockchain to start accepting subsequent votes. As the federal system of Switzerland encourages a decentralised administration, different voting periods are not prohibited for the same national election. An extension to the current voting protocol would allow for such a differentiation by attaching a regional identifier to open vote transactions $T_{Vo}$and voting transactions T_V. It would further be necessary to adjust the `SumCipherText Visitor` to account for these changes.

6.1.4 Voting Question

Although a secure electronic voting system is of absolute necessity, the integrity of the voting question itself is crucial as well. Tampering with its linguistic structure, such as negating or rephrasing it entirely, can affect the outcome of the election effectively. Thus, storing it on a distributed ledger can reduce such a risk to a great amount.

6.1.5 GHOST protocol

As the testing setup was communicating in near real time on the same host, network partitions did not occur. Thus, the GHOST protocol which resolves the canonical chain in case of forks was not absolutely required in the proof-of-concept voting system provided by this thesis' work. However, it is crucial that an implementation is provided for a distributed setup.

6.1.6 Multi-Way Elections and Limited Votes

Assumption A5 restricted this work's focus on binary votes only. Although empty votes can occur by voters who register to vote but never submit a decision to the voting network, voting for one or even multiple options is not yet implemented. However, an approach by [7] encodes a vote into a numeric representation which uses as base a number greater or equal to the size of the electorate. Then, each digit of such an encoded vote represents a single candidate. To calculate the final tally, the addition operation would be performed as usual. The digits of the resulting sum then represent the number of votes for each candidate. Correspondingly, the proofs would be required to operate on single digits of the encoded vote as well. To perform limited votes, [40] proposes to encode a vote as a vector of size $|c|$, the number of voting candidates. Computing the final tally as well as generating the proofs would then be done element-wise.

6.1.7 Human Perspicuity

Although a blockchain can distribute trust among different authorities, the cryptographic procedures to ensure a secure and integer vote are highly complex. Critics of electronic voting systems argue that a large part of the electorate will not be able to understand the verifiability properties in much detail [1]. Countermeasures, such as paper audit trails, are often generated after an electronic voting system has processed a vote, and require a certain degree of trust from the voter as well. Thus, future work needs to show whether a more human readable kind of verifiability can be constructed.

[1] E.g. `https://www.republik.ch/2018/03/08/e-voting-der-schweizer-sonderfall`

Bibliography

[1] Agora (2017). Agora: Bringing our voting systems into the 21st century.

[2] Agora (2018). Agora Official Statement Regarding Sierra Leone Election. `https://medium.com/agorablockchain/agora-official-statement-regarding-sierra-leone-election-7730d2d9de4e`. Accessed: 2018-10-10.

[3] Anane, R., Freeland, R., and Theodoropoulos, G. (2007). E-Voting Requirements and Implementation. *Proceedings - The 9th IEEE International Conference on E-Commerce Technology; The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services, CEC/EEE 2007*, pages 382–389.

[4] Anjum, A., Sporny, M., and Sill, A. (2017). Blockchain Standards for Compliance and Trust. *IEEE Cloud Computing*, 4(4):84–90.

[5] Avižienis, A., Laprie, J. C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

[6] Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., and Danezis, G. (2017). Consensus in the Age of Blockchains.

[7] Baudron, O., Fouque, P.-A., Pointcheval, D., Stern, J., and Poupard, G. (2001). Practical multi-candidate election system. *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing - PODC '01*, (august):274–283.

[8] Benaloh, J., Bernhard, M., Halderman, J. A., Rivest, R. L., Ryan, P. Y. A., Stark, P. B., Teague, V., Vora, P. L., and Wallach, D. S. (2017). Public Evidence from Secret Ballots.

[9] Beroggi, G., Moser, P., and Bierer, D. (2011). Evaluation der E-Voting Testphase im Kanton Zürich 2008-2011. Technical Report November, Statistisches Amt des Kantons Zürich.

[10] Blum, M., De Santis, A., Micali, S., and Persiano, G. (1991). Non-Interactive Zero Knowledge. *SIAM Journal on Computing*, 20(6):1084–1118.

[11] Bocek, T. and Stiller, B. (2018). Smart Contracts – Blockchains in the Wings. *Digital Marketplaces Unleashed*, pages 169–184.

[12] Brelle, A. and Truderung, T. (2017). Cast-as-Intended Mechanism with Return Codes Based on PETs.

[13] Buchman, E. (2016). Tendermint: Byzantine Fault Tolerance in the Age of Blockchains.

[14] Burge, J. E. and Brown, D. C. (2008). Software Engineering Using RATionale. *Journal of Systems and Software*, 81(3):395–413.

[15] Cachin, C. and Vukolić, M. (2017). Blockchain Consensus Protocols in the Wild.

[16] Castro, M., Castro, M., Liskov, B., and Liskov, B. (1999). Practical Byzantine fault tolerance. *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, pages 173–186.

[17] Chevallier-Mames, B., Fouque, P. A., Pointcheval, D., Stern, J., and Traoré, J. (2010). On some incompatible properties of voting schemes. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6000 LNCS:191–199.

[18] Cortier, V., Galindo, D., and Turuani, M. (2017). A formal analysis of the Neuchâtel e-voting protocol. Research report, Inria Nancy - Grand Est.

[19] Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2011). *Distributed Systems: Concepts and Design*, volume 5. Pearson.

[20] Cucurull, J., Guasch, S., and Galindo, D. (2016). Transitioning to a javascript voting client for remote online voting. In *SECRYPT*, pages 121–132.

[21] Damgard, I. (2002). On Σ-protocols. *Lecture notes for CPT*.

[22] De Angelis, S., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A., and Sassone, V. (2018). PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain. *CEUR Workshop Proceedings*, 2058:1–11.

[23] Die Bundesbehörden der Schweizerischen Eidgenossenschaft (2002). Bericht über den Vote électronique: Chancen, Risiken und Machbarkeit elektronischer Ausübung politischer Rechte. *Bundesblatt*, 154(5):645–700.

[24] Die Schweizerische Bundeskanzlei (2018). Vote électronique - Chronik. `https://www.bk.admin.ch/bk/de/home/politische-rechte/e-voting/chronik.html`. Accessed: 2018-10-08.

[25] Die Schweizerische Bundeskanzlei (vom 13. Dezember 2013 (Stand am 1. Juli 2018))). Verordnung der BK über die elektronische Stimmabgabe (VEleS) . `https://www.admin.ch/opc/de/classified-compilation/20132343/201807010000/161.116.pdf`. Accessed: 2018-10-08.

[26] Die Schweizerische Bundeskanzlei (vom 13. Mai 2015b). Umsetzung von Artikel 50 der Bundesverfassung. `https://www.admin.ch/opc/de/federal-gazette/2015/3881.pdf`. Accessed: 2018-07-11.

[27] Die Schweizerische Bundeskanzlei (vom 17. Dezember 1976 (Stand am 1. November 2015)a). Bundesgesetz über die politischen Rechte (BPR). `https://www.admin.ch/opc/de/classified-compilation/19760323/201511010000/161.1.pdf`. Accessed: 2018-07-11.

[28] Die Schweizerische Bundeskanzlei (vom 24. Mai 1978 (Stand am 15. Januar 2014)). Verordnung über die politischen Rechte (VPR). `https://www.admin.ch/opc/de/classified-compilation/19780105/201401150000/161.11.pdf`. Accessed: 2018-07-09.

[29] Douceur, J. R. (2002). The Sybil Attack. In *IPTPS '01 Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260.

[30] Elgamal, T. (1985). A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472.

[31] Escala, A., Guasch, S., Herranz, J., and Morillo, P. (2016). Universal cast-as-intended verifiability. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9604 LNCS:233–250.

[32] Fiat, A. and Shamir, A. (1987). How to prove yourself: Practical solutions to identification and signature problems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 263 LNCS:186–194.

[33] Galindo, D., Guasch, S., and Puiggalí, J. (2015). 2015 Neuchâtel's Cast-as-Intended Verification Mechanism. In Haenni, R., Koenig, R. E., and Wikström, D., editors, *E-Voting and Identity*, pages 3–18, Cham. Springer International Publishing.

[34] Gibson, J. P., Krimmer, R., Teague, V., and Pomares, J. (2016). A review of E-voting: the past, present and future. *Annales des Telecommunications/Annals of Telecommunications*, 71(7-8):279–286.

[35] Goldwasser, S., Micali, S., and Rackoff, C. (1989). The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, 18(1):186–208.

[36] Gritzalis, D. A. (2002). Principles and requirements for a secure e-voting system. *Computers and Security*, 21(6):539–556.

[37] Haenni, R., Koenig, R. E., Locher, P., and Dubuis, E. (2018). CHVote System Specification – Version 1.4.2. *IACR Cryptology ePrint Archive*, 2017:325.

[38] Hapsara, M., Imran, A., and Turner, T. (2017). E-voting in developing countries. In Krimmer, R., Volkamer, M., Barrat, J., Benaloh, J., Goodman, N., Ryan, P. Y. A., and Teague, V., editors, *Electronic Voting*, pages 36–55, Cham. Springer International Publishing.

[39] Hastings, N., Peralta, R., Popoveniuc, S., and Regenscheid, A. (2011). Security Considerations for Remote Electronic UOCAVA Voting.

[40] Hirt, M. (2010). Receipt-free K-out-of-L voting based on ElGamal encryption. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6000 LNCS, pages 64–82.

[41] Holman, M. R. and Lay, J. C. (2018). They See Dead People (Voting): Correcting Misperceptions about Voter Fraud in the 2016 U.S. Presidential Election. *Journal of Political Marketing*, 0(0):1–38.

[42] Internet Voting - Current Status of Internet Voting in the United States (2018). `https://www.verifiedvoting.org/resources/internet-voting/`. Accessed: 2018-10-09.

[43] Jonker, H., Mauw, S., and Pang, J. (2013). Privacy and Verifiability in Voting Systems. *Computer Science Review*, 10:1 – 30.

[44] Jonker, H. and Pang, J. (2011). Bulletin boards in voting systems: Modelling and measuring privacy. *Proceedings of the 2011 6th International Conference on Availability, Reliability and Security, ARES 2011*, pages 294–300.

[45] Korman, M. (2007). Secret-Ballot Electronic Voting Procedures Over the Internet.

[46] Kremer, S., Ryan, M., and Smyth, B. (2010). Election Verifiability in Electronic Voting Protocols. In *Computer Security – ESORICS 2010*, pages 389–404, Berlin, Heidelberg. Springer Berlin Heidelberg.

[47] Krimmer, R., Triessnig, S., and Volkamer, M. (2007). The development of remote e-voting around the world: A review of roads and directions. In Alkassar, A. and Volkamer, M., editors, *E-Voting and Identity*, pages 1–15, Berlin, Heidelberg. Springer Berlin Heidelberg.

[48] Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401.

[49] McCorry, P., Shahandashti, S. F., and Hao, F. (2017). A smart contract for boardroom voting with maximum voter privacy. In Kiayias, A., editor, *Financial Cryptography and Data Security*, pages 357–375, Cham. Springer International Publishing.

[50] Moran, T. and Naor, M. (2006). Receipt-Free Universally-Verifiable Voting with Everlasting Privacy. *Advances in Cryptology - CRYPTO 2006: 26th Annual International Cryptology Conference. Proceedings*, 4117:373–392.

[51] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

[52] OSCE/ODIHR (2013). *Handbook For the Observation of New Voting Technologies*. OSCE Office for Democratic Institutions and Human Rights (ODIHR).

[53] Rocket, T. (2018). Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies.

[54] Ryan, P. Y. A. and Teague, V. (2013). Pretty Good Democracy. In *Security Protocols XVII*, pages 111–130, Berlin, Heidelberg. Springer Berlin Heidelberg.

[55] Smyth, B. (2018). Ballot secrecy: Security definition, sufficient conditions, and analysis of Helios.

[56] Sompolinsky, Y. and Zohar, A. (2015). Secure high-rate transaction processing in bitcoin. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8975, pages 507–527.

[57] Standeskanzlei Graubünden (2015). Vote électronique – Auflösung des Consortiums. `https://www.gr.ch/DE/Medien/Mitteilungen/MMStaka/2015/Seiten/2015092101.aspx`. Accessed: 2018-10-08.

[58] State Electoral Office of Estonia (2017). General Framework of Electronic Voting and Implementation thereof at National Elections in Estonia.

[59] Staveley, E. (1972). *Greek and Roman Voting and Elections*. Aspects of Greek and Roman life. Cornell University Press.

[60] Syverson, P. (1994). A taxonomy of replay attacks [cryptographic protocols]. In *Proceedings The Computer Security Foundations Workshop VII*, pages 187–191.

[61] The Rust Programming Language (2014). `https://www.rust-lang.org/en-US/index.html`. Accessed: 2018-10-17.

[62] UN General Assembly (1948). Universal Declaration of Human Rights. `https://www.ohchr.org/EN/UDHR/Pages/Language.aspx?LangID=eng`. Accessed: 2018-10-08.

[63] Wu, Y. (2017). An E-voting System based on Blockchain and Ring Signature.

[64] Zheng, Z., Xie, S., Dai, H., Chen, X., and Wang, H. (2017). An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. *Proceedings - 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017*, pages 557–564.

Abbreviations

BFT	Byzantine Failure Tolerance
BP	Ballot Privacy
DPoS	Delegated Proof of Stake
DRE	Direct Recording Electronic Voting System
GHOST	Greedy Heaviest Observed Subtree
NIST	National Institute of Standardization and Technology
NIZKPK	Non-Interactive Zero-Knowledge Proof of Knowledge
PBB	Public Bulletin Board
PBFT	Practical Byzantine Failure Tolerance
PET	Plaintext-Equivalent-Tests
PGD	Pretty Good Democracy
PoA	Proof of Authority
PoS	Proof of Stake
PoW	Proof of Work
PVI	Public UCIV Information
SVI	Secret UCIV Information
UCIV	Universal Cast-as-Intended Verification

List of Figures

List of Tables

Appendix A

Installation Guidelines

The presented electronic voting system needs the following pre-requisites to be installed:

- **RustUp**[1] in order to install Rust and Cargo
- **Rust** in version 1.26.2 or above
- **Cargo** in version 1.26.0 or above

Then, follow the order below for initiating the blockchain network and vote using the corresponding client.

- Generate the required cryptographic material by changing your working into `generator_rs`. Invoke `cargo run -- keys` to generate a private-public keypair. Then, invoke `cargo run -- uciv 10 2` to generate UCIV information for ten voters and two voting options. Whereas the first command will generate a `public_key.json` and `private_key.json`, the second will create `public_uciv.json` and `public_uciv.json`.
- Create a genesis configuration file as follows:

```
  "version": "0.1.0",
  "clique": {
    "block_period": 15,
    "signer_limit": 2
  },
  "sealer": [
    "127.0.0.1:9000",
    "127.0.0.1:9001",
    "127.0.0.1:9002"
  ]
}
```

Listing A.1: genesis.json

[1] `https://rustup.rs/`

- Locate the **node_rs** directory and copy the created `genesis.json`, `public_key.json`, and `public_uciv.json` into its root folder.
- Change your working directory to **node_rs**.
- Start the first node by running `cargo run -- -v start -s -p 127.0.0.1:9000 127.0.0.1:3000`. This will compile the attached source code for your computer and then execute the compiled binary.
- Wait until the first block is mined and broadcast. According to the genesis file in Listing A.1, this will take 15 seconds.
- Then, open a new terminal window and start the second node by running `cargo run -- -v start -s -p 127.0.0.1:9001 127.0.0.1:3001`. Wait again 15 seconds until the second block is broadcast.
- In a new window, start the third node by running `cargo run -- -v start -s -p 127.0.0.1:9002 127.0.0.1:3002`.
- Now, locate the **client_rs** directory in a different terminal window. Copy the files `public_key.json`, `private_key.json`, `public_uciv.json`, and `public_uciv.json` into it.
- Open the vote by running `cargo run -- admin open 127.0.0.1:3000`.
- Submit a *yes* vote for the first voter by running `cargo run -- submit-vote yes 0 127.0.0.1:3001`.
- Submit a second *yes* vote for the second voter by running `cargo run -- submit-vote yes 1 127.0.0.1:3001`.
- Submit a *no* vote for the third voter by running `cargo run -- submit-vote no 2 127.0.0.1:3001`. Do the same for a fourth voter by running `cargo run - submit-vote no 3 127.0.0.1:3001`.
- Wait until the block which contains the transactions has been mined and broadcast.
- Close the vote by running `cargo run -- admin close 127.0.0.1:3000`. Wait until the block containing this transaction has been mined and broadcast.
- Fetch the final tally from any of the clients by running `cargo run -- count-votes 127.0.0.1:3001`. Make sure, the voting result is two supporting versus two opposing votes with a total of four votes.
- Eventually, stop all running nodes before creating a new election.

Appendix B

Contents of the CD

This work comes with an attached CD which contains the following items:

- A description of the CD's contents.
- All developed program sources, in particular `node_rs`, `client_rs`, `crypto_rs`, `generator_rs`, and `node_benchmark_rs`.
- This thesis in source files, PDF, and PostScript. Additionally, all of its contained figures in source format and PNG.
- The slide deck of the final presentation.

Appendix C

Benchmark Setup

Rust comes with its own benchmarking suite[1]. However, it is not yet considered stable and thus requires to have unstable features enabled. These are only available in nightly builds of Rust. Install a nightly build by running `rustup install nightly`. Then, benchmarks can be invoked by prefixing all `cargo` commands with `rustup run nightly` and providing the `--bench` flag.

The `node_benchmarking_rs` package was used to create the metrics shown in Figure 5.2. The numbers for Figure 5.1 were obtained by the benchmarking suite contained in `generator_rs`. Please refer to the packages' `README` for detailed operations on how to run the benchmarking suite.

All benchmarking metrics used in this work were obtained on an otherwise idle-running MacBook Pro Late 2013 with a 2.3 GHz Intel Core i7, 16 GB 1600 MHz DDR3 Memory, and a 500 GB Apple SSD for storage.

[1] `https://doc.rust-lang.org/1.8.0/book/benchmark-tests.html`