University of Zurich[UZH]

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

Design and Implementation of Blockchain-based E-Voting

Moritz Eck, Alex Scheitlin, Nik Zaugg
Switzerland
Student ID: 14-715-296, 15-709-959, 12-716-734

Supervisor: Christian Killer, Bruno Rodrigues
Date of Submission: February 17, 2020

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

ifi

Master Project 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

Abstimmungen sind ein wesentlicher Bestandteil moderner Demokratien. Daher sind Veränderungen am Prozess eine heikle Angelegenheit und werden kontrovers diskutiert. Ein Paradebeispiel dafür ist die Einführung der elektronischen Fernabstimmung (E-Voting), welche als Ergänzung oder sogar als Ersatz für die traditionelle Wahlurne oder die briefliche Abstimmung eingesetzt werden kann. Um die Vertraulichkeit der Abstimmung und die Überprüfbarkeit des Verfahrens zu gewährleisten, bestehen hohe Anforderungen an solche Systeme. In den letzten Jahren wurde viel Forschungsarbeit geleistet, wie man kryptographische Techniken anwenden kann, um diesen Anforderungen gerecht zu werden. Einige Länder haben bereits zentralisierte Systeme zur elektronischen Stimmabgabe getestet oder verwenden diese aktiv. Die jüngsten Entwicklungen im Bereich der Blockchain-Technologie bieten geeignete Lösungen zur Dezentralisierung solcher Systeme. Aufbauend auf etablierten kryptographischen Techniken schlägt diese Arbeit ein sicheres und verifizierbares Design für ein verteiltes E-Voting System vor. Die Ver- und Entschlüsselungsschlüssel werden auf verteilte Weise von mehreren kooperierenden Parteien generiert. Die Verschlüsselung der Stimme und der Beweis deren Korrekt- und Unversehrtheit werden auf dem Gerät des Wählers erzeugt. Schließlich werden die verschlüsselten Stimmen öffentlich auf dem dezentralisierten Register (*d.h.* auf der Blockchain) verifiziert und gespeichert. Basierend auf einer Blockchain wird das entworfene Design in einem Prototypen umgesetzt. Das System ist verifizierbar und die Privatsphäre des Wählers wird durch kryptographische Grundsätze gewährleistet. Zusätzlich werden die Einschränkungen und zukünftigen Anwendungen von Blockchain im Zusammenhang mit der E-Voting hervorgehoben.

Abstract

Voting is an essential part of modern democracies. Thus, changes to the processes are always delicate and highly debated. A prime example of this is the introduction of remote electronic voting (E-Voting) which can be deployed as a supplement or even replacement for traditional ballot-box or postal voting. There exist high requirements for such systems to satisfy the vote's privacy and the process' verifiability. In recent years, a lot of research has been conducted on how to apply cryptographic techniques to meet these requirements. Some countries have even already tested or are using centralized E-voting systems. The recent developments in the blockchain area provide suitable solutions to decentralize such systems. Built on established cryptographic techniques, this work proposes a secure and verifiable design for a distributed E-voting system. The encryption and decryption keys are generated in a distributed manner by multiple cooperating parties. The encryption of the vote and the proof for the vote's validity are generated on the voter's device. Finally, the encrypted votes are publicly stored and verified on the decentralized public ledger (*i.e.*, the blockchain). The proposed design is implemented in a proof-of-concept E-Voting system based on a blockchain. The system is verifiable and the voter's privacy is ensured through cryptographic primitives. Additionally, limitations and future applications of blockchains in the context of E-Voting are highlighted.

Contents

Chapter 1

Introduction

Fair and secure votes are one of the cornerstones of democracies. Vote-buying, voter manipulation or just simple human error are ever-present and skew voting results and tamper with people's trust in such systems. Since the eighties, many researchers have been looking for alternative solutions involving electronic means [14]. Although research suggests that remote E-Voting (REV; or interchangeably just E-Voting) does not necessarily increase voter turnout [10, 13], REV can still be beneficial for people with disabilities, or people living abroad or in remote places, to participate. This research area is vast and complex, involving many disciplines from cryptography, security as well as legal aspects. Moreover, the lack of acceptance and trust of citizens into E-Voting systems poses additional obstacles [30]. The main challenge of E-Voting systems is to find an acceptable balance between privacy (the voter's choice cannot be linked to the voter) and verifiability (the voter can be sure that his or her vote was included in the ballot as intended) [33, 34]. Over the recent years, a lot of research has emerged that contributed to and addressed newly refined notions of privacy and verifiability. Still, practical implementations of such a system remain complex and are therefore seldom realised. One example often mentioned is Estonia, which allows its citizens to vote electronically on nation-wide elections [40]. A common aspect of recent E-Voting systems is their trust in central authorities with central control. Although the system might be distributed or replicated, trust is still centralized. Luckily, the emerging blockchain technology seems to offer a solution to this problem - a decentralized, tamper-proof and append-only ledger. Still, until now only a few works combine electronic voting with distributed ledger technologies [31, 39, 42]. In 2018, Provotum [41] was developed; a proof-of-concept E-Voting system using a private proof-of-authority Ethereum blockchain. Although the system showcases a fully functioning E-Voting system promising a high degree of verifiability and privacy, it is not without flaws. The system relies on a centralized server to handle encryption and proof verification. This leads to the following problems:

i **Single Point of Failure/Single Source of Trust**: As only the central server can decrypt and verify votes, all participants in the system need to trust the central authority which defeats the initial purpose of the decentralization with a blockchain.

ii **Server-side Cryptography/Proof Verification**: The cryptographic operations all occur on a central server. They are also not verified or stored on the blockchain,

adding no possibility for verification by voters and third parties. Instead of trusting a central authority, the encryption and proof generation needs to happen on the client-side to increase the voters' trust.

1.1 Project Goals and Contributions

The goal of this work is to overcome the aforementioned shortcomings. To this end, a new proof-of-concept E-Voting system is designed and implemented. The system will use a private proof-of-authority blockchain to act as the secure public bulletin board (i). Furthermore, client-side vote encryption and proof generation are employed (ii). All votes and proofs are stored and verified inside a smart contract [9] on the blockchain (i). The system should strive for the best compromise between maximizing the verifiability while retaining as much privacy as possible. To achieve this goal, distributed key generation, zero-knowledge proofs, and homomorphic encryption will be used as primary cryptographic primitives. The following work describes the steps that were necessary to achieve the objective and further outlines the required theoretical building blocks to engineer the final architecture. All the code created in the project is published under the MIT license and freely available on Github[1][2].

1.2 Outline

In chapter 2, the theoretical building blocks and cryptographic primitives are denoted. Then, in chapter 3, the related work is discussed. Subsequently, in chapter 4, the design of the E-Voting system and the different stakeholders are introduced. Chapter 5 describes the voting process itself and chapter 6 evaluates the proposed system. Finally, chapter 7 defines future work and concludes this project.

[1] E-Voting System: `http://bcbev.ch/provotum-v2`
[2] Cryptographic Library: `http://bcbev.ch/provotum-v2-crypto`

Chapter 2

Background

This chapter discusses the different technical aspects and concepts required to understand how E-Voting systems work. The first part introduces cryptographic concepts that are used (i) to encrypt votes to preserve privacy and (ii) to generate proofs (*e.g.*, for encryption and decryption) to provide verifiability. Further, it is shown how the cryptographic system can be used in a distributed, untrusted setting. The second part introduces blockchain technology, its properties and how blockchains are applicable in the context of E-Voting systems.

2.1 Cryptographic Building Blocks

This section covers the cryptographic building blocks used to establish privacy and verifiability in E-Voting systems. The concepts and techniques used in this work are described from a technical standpoint of view.

2.1.1 Cryptographic Hash Functions

In general, hash functions are used to generate a "fingerprint" of some piece of data [55] and can be defined as follows:

> *"A function $h : X \to Y$ is a hash function, if $|X| \gg |Y|$ and h(x) can be computed efficiently for all $x \in X$".* [46]

Moreover, the function h takes an input x of arbitrary size and produces an output y of fixed (usually much smaller) size, the hash value [46, 55]. In cryptography, hash functions have specific (security) properties [46]:

- *Preimage Resistance (one-way)*: The computation of the hash value can be done efficiently but it is infeasible to invert the function (*i.e.*, find x for a given y such that $h(x) = y$) [55].

- *Second Preimage Resistance*: Given an input x, it is computationally infeasible to find a different input $z \neq x$ that maps to the same hash value (*i.e.*, $h(x) = h(z)$) [55].
- *Collision Resistance*: It is computationally infeasible to find any pair (x, z) such that $h(x) = h(z)$ [55].

A cryptographic hash function has to be (i) hard to invert (*i.e.*, the hash function is preimage resistant) and (ii) it must be impossible to find a collision (*i.e.*, the hash function is either second preimage resistant or collision-resistant) [46]. Given these properties hash functions can be used for various applications such as verifiability proofs (*i.e.*, (non-interactive) zero-knowledge proofs of knowledge, described in 2.1.5). Examples of such are proofs that verify that an encrypted vote is valid (*i.e.*, represents a supporting "yes" or rejecting "no" vote) or that the decryption was done properly.

2.1.2 Public-Key Cryptography

Public-key cryptography is asymmetric, meaning there are two separate keys involved [55]. A public-key encryption scheme consists of the following components: a plaintext p to encrypt, an encryption algorithm E, a key pair with a public key k_{pub} and a private key k_{priv}, an encrypted ciphertext c and a decryption algorithm D [55]. While the encryption of plaintext can be done by anyone by using the public key, the decryption of ciphertext can only be done by using the private key [46]. Following, the relation between the mentioned components is illustrated:

$$E(p, k_{pub}) = c \quad \rightarrow c \rightarrow \quad D(c, k_{priv}) = p$$

Further, the generation of the key pair and the encryption and decryption of the data with the respective keys need to be computationally easy. However, inferring the private key from the public key or the decrypted data from the encrypted data and the public key needs to be computationally infeasible [55].

Public-key cryptography is widely used and applied for different purposes. It is also the foundation for the ElGamal cryptosystem (described in 2.1.3) which can be used to encrypt votes in E-Voting systems.

2.1.3 ElGamal Cryptosystem

The ElGamal cryptosystem is a public-key cryptosystem defined over cyclic groups G and is based on the difficulty of solving the discrete logarithm problem in G. It uses a cyclic group Z_p^*, a multiplicative group G^* defined over a finite field of integers Z_p (*i.e.*, multiplications of integers are always modulo p). For security reasons, the finite field with generator g and order of the group q is required to be a prime field (*i.e.*, both p and q are prime numbers and p should be at least 2048 bits large). The reason it can be used to encrypt votes in an E-Voting system is its additive homomorphic property (described in 2.1.4) which allows counting votes without requiring to decrypt them first [25].

Key Pair Generation. The private key sk is a secure random value in the range $[1, q-1]$. The public key pk is the generator g of the chosen cyclic group Z_p^* to the power of the private key modulo p (2.1).

$$(sk, pk) = (r, h) = (r, g^r \pmod p) \tag{2.1}$$

Encryption. To encrypt a plaintext m (*i.e.*, a no vote of value 0 or a yes vote of value 1) into a ciphertext (c_1, c_2), a secure random value r in the range of $[1, q-1]$ needs to be drawn. The cipher's first component c_1 (2.2) is equal to the group's generator g to the power of the random value r modulo p. The second component c_2 (2.2) is the product of the public key h to the power of the random value r and the plaintext (*i.e.*, vote) $m \in \{0, 1\}$ again modulo p.

$$(c_1, c_2) = (g^r \pmod p, h^r m \pmod p) \text{ where } m \in \{0, 1\} \tag{2.2}$$

Decryption. To decrypt a ciphertext (c_1, c_2) the multiplicative inverse of the cipher's first component c_1 to the power of the private key sk is multiplied with the cipher's second component c_2 modulo p (2.3).

$$m = (c_1^{sk})^{-1} c_2 \pmod p \tag{2.3}$$

2.1.4 Homomorphic Encryption

The idea of homomorphic encryption is to operate on encrypted data such that its decrypted result is the same as if the operation had been performed on the original plaintext data. Thus, confidential data can remain private while still being able to be processed in an untrusted environment (*e.g.*, a public cloud provider) [4, 28].

The functionality is based on a homomorphism (*i.e.*, a structure-preserving mapping) existing between a plaintext and a ciphertext operation which can be defined with the equality in (2.4).

$$e_{key}(m_1) \otimes e_{key}(m_2) = e_{key}(m_1 \oplus m_2) \tag{2.4}$$

The equality in (2.4) shows that using homomorphic encryption two encrypted messages can be combined into one without the use of an encryption key. The operators $\oplus$ and $\otimes$ represent arbitrary operations that depend on the implementation of the cryptosystem. This property can be used to tally votes in an E-Voting system. Every voter V can encrypt his vote $vote_V$ using the public key pk, resulting in the encrypted vote: $e_{pk}(vote_V)$ [33].

Using the equality defined in (2.4), all votes can be added without having to decrypt the individual votes first. The encrypted result looks as follows in (2.5):

$$e_{pk}(vote_1) \otimes ... \otimes e_{pk}(vote_n) = e_{pk}(vote_1 \oplus ... \oplus vote_n) \tag{2.5}$$

To make the previously described ElGamal cryptosystem (2.1.3) additively homomorphic, both the encryption and decryption algorithms need to be slightly adjusted. During the encryption step (2.2), the plaintext m needs to be encoded by using the generator g to the power of m instead of just using m (2.6). After the decryption step (2.3), the result needs to be decoded either using a brute force algorithm (*i.e.*, by probing all possible values) or algorithms like the *baby-step giant-step* to compute the discrete logarithm of g^m. The equations (2.7) and (2.8) show how two ciphertexts $E(m_1)$ and $E(m_2)$ are multiplied by adding the components exponents producing the same result as if the plaintexts m_1 and m_2 were added.

$$E(m) = (c_1, c_2) = (g^r, h^r * g^m) \tag{2.6}$$
$$E(m_1) * E(m_2) = (g^{r_1+r_2}, h^{r_1+r_2} * g^{m_1+m_2}) = E(m_1 + m_2) \tag{2.7}$$
$$E(m_3) = E(m_1 + m_2) = (E(m_1)_{c_1} * E(m_2)_{c_1}, E(m_1)_{c_2} * E(m_2)_{c_2}) \tag{2.8}$$

2.1.5 Zero-Knowledge Proof

A zero-knowledge proof (ZKP) is a cryptographic method by which a prover P can prove to a verifier V that it knows about a secret s (e.g., the content of a vote) without revealing anything about s apart from the fact that it knows s. The proof is called zero-knowledge because V learns nothing about s apart from the fact that P knows about s [33, 54].

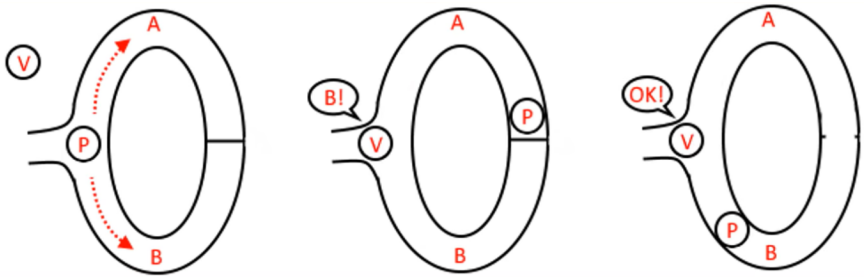

Figure 2.1: Ali Baba's Cave

Interactive ZKPs are best explained using the Ali Baba's cave example[1] introduced by Quisquater et al. [50]. The cave has a single entry, is shaped like a ring with two paths (A and B) and contains a hidden door separating the two paths. P knows the password to open the hidden door but does not want to reveal it. To prove to V that P knows the password, P enters the cave and chooses a path. Next, V enters the cave and demands that P exits the cave using path A or B, chosen at random. Since P knows the password,

[1] `https://www.youtube.com/watch?v=0Sy6nb72gCk`

it is trivial for it to pass the test. On the other hand, had P not known the password, he would have only had a 50% chance of returning from the demanded path. For V to increase its trust, it can challenge P multiple times thereby decreasing the probability of P being a fraudulent prover by a factor of two every time (see Equation 2.9). Therefore, V can repeat this process as many times as deemed necessary until the desired level of confidence is obtained [33, 50, 54].

$$p_{cheat} = \sum_{i=1}^{n} \frac{1}{2^i} \tag{2.9}$$

In E-Voting systems, ZKPs can be used to submit a proof of a vote's validity when a voter casts its vote. This ensures that only valid votes are considered in the final tally [33, 54]. Moreover, ZKPs can also be used to prove that the decryption was done using the private key associated with the public key previously used for encryption.

The issue with interactive ZKPs is that the communication between the prover and verifier is time-consuming, costly and often not feasible in practice. An alternative is a **non-interactive ZKP (NIZKP)** which is a ZKP that does not require the interaction between the prover and the verifier. The interaction (*i.e.*, the challenge chosen by the verifier) is replaced by a cryptographic hash function serving as a random oracle. This is also known as applying the **Fiat-Shamir heuristic** [27] to an interactive ZKP. The following proofs are ZKPs transformed to NIZKP using the Fiat-Shamir heuristic. Thus, they can be used in E-Voting systems without the need for interaction to generate and validate the proofs.

Schnorr Proof - Key Generation

To show knowledge of an ElGamal private key r that belongs to a public key h (*i.e.*, $h = g^r$), the Schnorr Proof [52] can be used. It is a proof of knowledge of a discrete logarithm of $r = log_g(g^r)$ [26].

Proof Generation. The proof can be created as shown in (2.10). The private/public key pair (r, h) is needed as input. Additionally, a secure random value a in the range $[1, q-1]$ is needed. $q = \frac{p-1}{2}$ is the additive modulus of the cyclic group Z_q^+ over the field F_q (*i.e.*, the cyclic group in which all additive operations take place) and g is the group's generator. The id is the unique identifier of the prover. To generate the proof, first, the commitment b, the generator g to the power of a, is hashed together with the unique id and the public key h. This forms the challenge c that is the first part of the proof. The second part of the proof is the response, which is the sum of the random value a and the product of the challenge c and the private key r.

(2.10)

input	keypair $= (sk, pk) = (r, h)$
commitment	$b = g^a \pmod p$
challenge	$c = hash(id, h, b) \pmod q$
response	$d = a + c \times r \pmod q$
output	proof $= (c, d)$

Proof Verification. To verify the Schnorr proof (c, d), first, the commitment b' and challenge c' need to be recomputed as shown in (2.11). Then, the recomputed challenge c' needs to match the original challenge c. And the generator g to the power of the response d needs to be equal to the recomputed commitment b' multiplied by the public key h to the power of the original challenge c. If both checks are successful, the proof is deemed valid.

(2.11)

input	proof $= (c, d)$
recompute commitment	$b' = g^d / h^c \pmod p$
recompute challenge	$c' = hash(id, h, b') \pmod q$
compare the hashes	$c \stackrel{!}{=} c'$
verify the proof	$g^d \pmod p \stackrel{!}{=} b' \times h^c \pmod p$

Chaum-Pedersen Proof - Decryption

The Chaum-Pedersen proof [17] can be used to affirm the correct decryption of a cipher (c_1, c_2) using the private key r. It is a proof of discrete logarithm equality of $log_g(g^r) = log_h(h^r)$ and is similar to the Schnorr Proof [19].

Proof Generation. To generate the proof for a ciphertext (c_1, c_2), as shown in (2.12), a secure random value x in the range $[1, q-1]$ is needed. The commitment consists of two parts a and b. a is equal to the cipher's first component c_1 to the power of the random value x and b is equal to the generator g to the power of x. The challenge c is computed by hashing the unique prover id, the cipher's components c_1 and c_2, and the commitments components a and b using a cryptographic hash function. The final response consists of f and d. f is computed by adding the random value x to the product of the challenge c and the private key r and d is the cipher's first component c_1 to the power of the private key r. The proof itself consists of the commitment (a, b) and the response (f, d).

(2.12)

$$\begin{aligned}
\textbf{input} \quad & \text{ciphertext} = (c_1, c_2) \\
\textbf{commitment} \quad & a = c_1^x \pmod p \\
& b = g^x \pmod p \\
\textbf{challenge} \quad & c = hash(id, c_1, c_2, a, b) \pmod q \\
\textbf{response} \quad & f = x + c \times r \pmod q \\
& d = c_1^r \pmod p \\
\textbf{output} \quad & \text{proof} = (a, b, f, d)
\end{aligned}$$

Proof Verification. The verification of a chaum-pedersen proof, consisting of the commitment (a, b) and the response (f, d), for a ciphertext (c_1, c_2) is presented in (2.13). First, the challenge c' is recomputed by hashing the prover's unique id together with the cipher (c_1, c_2) and the response (a, b) extracted from the proof. To verify the proof's validity, two checks are performed. The cipher's first component c_1 to the power of the response's fist component f needs to be equal to the commitment's first component a multiplied with the response's second component d to the power of the recomputed challenge c'. Finally, it is checked whether the generator g to the power of the response's fist component f is equal to the commitment's second component b multiplied with the public key h to the power of the recomputed challenge c'.

(2.13)

$$\begin{aligned}
\textbf{input} \quad & \text{proof} = (a, b, f, d) \\
& \text{ciphertext} = (c_1, c_2) \\
\textbf{recompute challenge} \quad & c' = hash(id, c_1, c_2, a, b) \pmod q \\
\textbf{verify proof} \quad & c_1^f \pmod p \stackrel{!}{=} a \times d^{c'} \pmod p \\
& g^f \pmod p \stackrel{!}{=} b \times h^{c'} \pmod p
\end{aligned}$$

Disjunctive Chaum-Pedersen Proof - Encryption

The disjunctive chaum-pedersen proof is a modification of the chaum-pedersen proof introduced before [6]. This type of ZKP is also called OR-proof or membership proof since the vote contained in the ballot is proven to be either a no-vote (value 0) or a yes-vote (value 1) and guaranteed to be nothing else. The proof does this without revealing the actual content of the vote.

Proof Generation. The proof generation is always composed of two parts. The yes-part and the no-part for which only ever one part exists (*i.e.*, the knowledge of the actual vote and in both cases the missing part needs to be simulated). For example, if the ballot

contains a yes-vote, the no-vote part of the proof needs to be simulated. The proof is generated as shown in (2.14) which is an example for a yes-vote proof whereby c_0, f_0, x are secure random values in the range $[1, q-1]$ and r is the random value used in the encryption of the vote.

(2.14)

input	ciphertext $= (a, b)$
simulation (v_0 - part)	$(a_0, b_0) = (g^{f_0}/a^{c_0} \pmod p, h^{f_0}/b^{c_0} \pmod p)$
proof (v_1 - part)	$(a_1, b_1) = (g^x \pmod p, h^x \pmod p)$
challenge	$c = hash(id, a, b, a_0, b_0, a_1, b_1) \pmod q$
	$c_1 = c - c_0 \pmod q$
response	$f_1 = x + c_1 \times r \pmod q$
output	yes-vote proof $= (a_0, b_0, a_1, b_1, c_0, c_1, f_0, f_1)$

In the case of the no-vote proof (2.15), instead of simulating c_0, f_0 the values c_1, f_1 (secure random values in the range $[1, q-1]$) are simulated. Another slight change is that instead of using b in the simulation, b/g is used. Actually, in the yes-vote proof, b/g^0 is taken which is equal to b. In the no-vote proof, where the yes-vote (of value 1) is simulated, b/g^1 is taken [6].

(2.15)

input	ciphertext $= (a, b)$
simulation (v_1 - part)	$(a_1, b_1) = (g^{f_1}/a^{c_1} \pmod p, h^{f_1}/(b/g)^{c_1} \pmod p)$
proof (v_0 - part)	$(a_0, b_0) = (g^x \pmod p, h^x \pmod p)$
challenge	$c = hash(id, a, b, a_0, b_0, a_1, b_1) \pmod q$
	$c_0 = c - c_1 \pmod q$
response	$f_0 = x + c_0 \times r \pmod q$
output	no-vote proof $= (a_0, b_0, a_1, b_1, c_0, c_1, f_0, f_1)$

Proof Verification. Both the yes- and no-vote proof are validated using the algorithm described in (2.16). After the challenge c' is recomputed, various checks are needed to validate the proof.

(2.16)

$$\begin{aligned}
\textbf{input} \qquad & \text{vote} = (a, b) \\
& \text{proof} = (a_0, b_0, a_1, b_1, c_0, c_1, f_0, f_1) \\
\textbf{recompute challenge} \qquad & c' = hash(id, a, b, a_0, b_0, a_1, b_1) \pmod q \\
\textbf{compare the hashes} \qquad & c_0 + c_1 \pmod q \stackrel{!}{=} c' \\
\textbf{verify the proof} \qquad & g^{f_0} \pmod p \stackrel{!}{=} a_0 \times a^{c_0} \pmod p \\
& g^{f_1} \pmod p \stackrel{!}{=} a_1 \times a^{c_1} \pmod p \\
& h^{f_0} \pmod p \stackrel{!}{=} b_0 \times b^{c_0} \pmod p \\
& h^{f_1} \pmod p \stackrel{!}{=} b_1 \times (b/g)^{c_1} \pmod p
\end{aligned}$$

2.1.6 Multiparty Computation (Cooperative Decryption)

In the case of E-Voting, the government is often considered the trusted authority which holds the private key to decrypt the ballots. Since voting is highly sensitive, it makes sense to split the access to the private key among several parties (*e.g.*, in the case of Switzerland, among the cantons). By doing this, a single point of failure is avoided as otherwise compromising a single entity leads to a break in the E-Voting system's security. An example of such an implementation is described next [33, 54].

Distributed Key Generation and Decryption

The following algorithms used for the distributed key generation (DKG) and decryption introduces a threshold cryptosystem where n out of n parties need to be present to decrypt a ciphertext [19, 48]. It can be used in conjunction with the ElGamal cryptosystem described in 2.1.3. However, the key generation and decryption steps need to be adjusted as follows. The encryption remains the same as in (2.2).

Key Pair Generation. The key pair is created similarly to (2.1). Each party $i \in [1, n]$ creates a key pair (sk_i, pk_i) as shown in (2.17) resulting in n key pairs. While the private keys sk_i are kept secret, the public keys pk_i are collected and combined to form the system's public key $pk = h$ (2.18) which will then be used to encrypt the votes.

$$(sk_i, pk_i) = (r_i, h_i) = (r_i, g_i^r \pmod p) \tag{2.17}$$

$$pk = h = \prod_{i=1}^{n} pk_i \pmod p \tag{2.18}$$

Decryption. The decryption is done in two steps. First, each party decrypts the ciphertext (c_1, c_2) using its private key sk_i producing a decrypted share d_i (2.19). The shares themselves do not reveal anything about the plaintext. Then, the decrypted shares are collected and combined to reveal the plaintext (2.20). Since all ballots are encoded (*i.e.*, vote to the power of the generator g^m) for them to be additively homomorphic, the result first needs to be decoded as described in paragraph 2.1.4. Note that the decryption involves computing the multiplicative inverse.

$$d_i = c_1^{sk_i} \pmod{p} \tag{2.19}$$

(2.20)

$$\begin{aligned}
\textbf{input} \qquad & \text{ciphertext} = (c_1, c_2) \\
& \text{decrypted shares} = [d_1, ..., d_i, ..., d_n] \\
\textbf{decryption} \qquad & m = \frac{c_2}{\prod_{i=1}^{n} d_i} \pmod{p} \\
& = c_2 \times (\prod_{i=1}^{n} d_i)^{-1} \pmod{p}
\end{aligned}$$

2.2 Blockchain Technology

A blockchain can best be described as a distributed, append-only database. This database consists of blocks of transactions that need to be validated and linked together in a chain, hence the name blockchain. A block is a type of data structure that bundles together transactions on the blockchain network. Each of these blocks is linked by a cryptographic hash to the previous block. When transactions are exchanged and synchronized across the network, the state of the blockchain changes. These transactions can be financial (*e.g.*, Bitcoin) or of a generic type such as Smart Contracts (*e.g.*, Ethereum), that allow the execution of arbitrary code on the blockchain.

2.2.1 Participants

Generally, blockchain networks consist of two types of participants: (i) miners and (ii) users.

Miners. Miners are nodes within the network that help grow the blockchain and perform the consensus protocol and transaction validation. Furthermore, they are responsible for bundling valid transactions into blocks as well as appending them to the blockchain.

Users. The users interact with the blockchain, can create transactions and send them to the network for miners to validate. Additionally, users might also read and analyse the state of the blockchain without altering its state.

To interact with the blockchain (send and receive transactions), miners and users create blockchain accounts, public/private key pairs also referred to as **wallets**.

2.2.2 Permissioned vs. Permissionless

Generally, there exist two distinct ways in which a blockchain can be set up: (i) permissioned and (ii) permissionless. The following two paragraphs describe their main differences.

Permissioned Blockchains. In a permissioned blockchain, a central authority grants write/read access to specific entities within the network. Further distinctions can be made depending on whether the blockchain is **public-permissioned** (*i.e.*, reading is possible for everyone) or **private-permissioned** (*i.e.*, reading is restricted to selected entities). Popular examples of permissioned blockchains are Hyperledger Fabric[2] or Corda[3].

Permissionless Blockchains. Permissionless blockchains represent the majority of current blockchains, such as Bitcoin [43] and Ethereum [57]. In this form, anyone can join and leave as a writer/reader at any time. There does not exist a central authority that manages the participants of the network. The blockchain is open to be read by anyone.

2.2.3 Consensus

Distributed systems, such as blockchains, need a way to agree upon the next state of the network. Traditional consensus mechanisms included Byzantine Agreement Protocols [47] which were susceptible to Sybil Attacks [45]. Therefore, blockchains need different consensus mechanisms to decide which blocks to add to the blockchain.

Proof of Work (PoW). This consensus mechanism is mostly used by public permissionless blockchains such as Bitcoin or Ethereum. In PoW, the miners have to solve a difficult mathematical problem (mining), allowing them to add the next block to the chain, should they solve it correctly. The difficulty of the mathematical problem is adjusted such that blocks are produced in a fixed time interval. A large drawback of PoW is the necessary "hash power" of the miners, which in turn uses a lot of electricity. At the time of writing, the whole Bitcoin network consumes more energy than Switzerland or Austria according to an index created at Cambridge University[4].

[2]https://www.hyperledger.org/projects/fabric
[3]https://www.corda.net
[4]https://www.cbeci.org/comparisons

Proof of Authority (PoA). PoA [3] does not depend on nodes solving arbitrarily difficult mathematical problems but instead uses a set of *authorities i.e.*, nodes that are explicitly allowed to create new blocks and secure the blockchain. These authorities are usually referred to as *sealers* or *validators* of the blockchain. The new state of the chain has to be signed off by the majority of authorities, in which case it becomes a part of the permanent record in the form of an additional block. This setup has the advantage that it is a lot less computationally intensive than PoW and that blocks can be created more predictably (*i.e.*, in more steady time intervals).

This work focuses on E-Voting in a setting where the authorities are known beforehand and can be trusted. These authorities will form the validators on the PoA blockchain.

2.2.4 Useful Properties

The following paragraphs highlight some of the main properties of blockchains with E-Voting in mind [58].

Public Verifiability. As blockchains have a distributed nature and blocks are validated and added by miners, every participant of the network will eventually have the same view of the network. Blockchains use cryptographic hashes. Every new block contains the hash of the previous block and, therefore, forming a chain where each new block reinforces the integrity of the preceding block. It ensures that the data in the previous block must have existed then. Thus, anyone can verify that new blocks were added correctly through the applied consensus mechanism. This means that anyone can verify the correctness of the blockchain state.

Immutability. Once a transaction has been written to the blockchain, it is practically impossible to alter or remove it. In the case where reversing a transaction is desirable, it must be done by executing a separate transaction and, therefore, both transactions remain visible.

Redundancy. In distributed systems, redundancy is required in many cases. Blockchain systems inherently possess this feature as each miner stores a replica of the whole blockchain on its machine.

2.2.5 Smart Contracts

The term *smart contract* was first coined by Nick Szabo in 1994 [56] and describes a computerized transaction protocol that executes predefined terms of a contract. This minimizes the risk of exceptions and removes the need for trusted intermediaries. But these contracts need suitable infrastructure to be considered ”smart”. Since the introduction of Bitcoin in 2009, smart contracts became more popular. Although Bitcoin offers only a

non-turing-complete scripting language, still, complex transactions can be scripted such as escrow contracts or multi-signature contracts. Ethereum introduced a turing-complete scripting language, which allows for general purpose applications. All transactions of a contract are run inside the Ethereum Virtual Machine (EVM) and each transaction has to be paid for to prevent Denial of Service attacks (DoS).

2.2.6 Blockchain and E-Voting

Most E-Voting systems make use of a secure public bulletin board (SPBB) to provide a publicly verifiable log of communication of the ongoing election or vote and to store the final result [33, 35]. In Jonker et al. [34], the main properties that such a SPBB should have are as follows: (i) anyone can read what is on the board, (ii) information on the board cannot be altered and (iii) everyone sees the same state of the board.

Considering these properties, blockchain technology offers suitable characteristics. As described in 2.2.4, blockchains can be described as a tamper-proof, distributed and public ledger.

Switzerland requires by law that votes be verifiable (see chapter 3) for any E-Voting system in Switzerland to be used by the majority of the public. In that sense, a public permissioned blockchain is most suitable to be used as the SPBB. The advantage of using a blockchain over an always online trusted third party (TTP) is that the authority does not need to be trusted with the full power over the recording and tallying of the votes. By doing so, there no longer exists a single point of failure. Furthermore, additional validator roles could be assigned to NGOs, states or any other partially trusted organization to further increase trust in the network [58].

Chapter 3

Related Work

The research revolving around voting and E-Voting is an important and active area of research. Started by the work of Chaum [14] about secure messaging, a plethora of subsequent research efforts were made, many of which are summarized and discussed in recent survey papers [1, 2, 7, 12, 33, 54]. The usage of electronic voting (E-Voting) systems is controversially discussed in society. For some, it is simply an extension to concepts such as e-commerce and e-banking whereas for others it is a huge security risk and poses a threat to a country's democratic system [24].

3.1 Properties of Electronic Voting

Initial work in this field focused on ballot secrecy (*i.e.*, keeping the voter's ballot secret) while additionally allowing the voter a way to trace the vote in the system (*i.e.*, verifiability) [14]. These two dimensions - privacy and verifiability - are ever-present in the realm of E-Voting.

Vote Privacy. Keeping the vote private is recognized as a fundamental human right as also stated in the Federal Constitution of the Swiss Confederation (Article 34.2) [22] and the Universal Declaration of Human Rights [5]. In other terms, vote secrecy or ballot secrecy is the requirement that cast votes must remain secret at all times and cannot be linked back to the voter. Interestingly, initial work in the field such as in [16] issued receipts upon voting, allowing the voter to trace the vote on the bulletin board. The work by Benaloh and Tuinstra [8] then showed that this strategy is flawed and that the issued receipts facilitate vote-buying and coercion. Their work introduced the notion of *receipt-freeness*. Many subsequent systems were directly influenced by this new notion [32, 44, 51] and completely abandoned the idea of receipts. In [33, 35], it is argued that *receipt-freeness* alone does not suffice to hinder a more powerful coercer to manipulate a voter's choice. A more powerful coercer may still hinder the voter from voting, obtain the voter's credentials or force the voter to vote randomly. These observations introduced the notion of *coercion-resistance* defined below. To guarantee *receipt-free* and

coercion-resistant systems, cryptographic schemes are employed such as homomorphic encryption [1], zero-knowledge proofs, mix nets, re-encryption and blind signatures [33]. Current research agrees upon the following key properties regarding vote privacy:

- **Ballot Secrecy (BS)**: cast votes must remain secret at all times and cannot be linked back to the voter
- **Receipt-Freeness (RF)**: a voter must not be able to prove to any third party that she has cast a particular vote
- **Coercion-Resistance (CR)**: extends the notion of *receipt-freeness* by ensuring protection against forced abstention, randomized voting or giving up voter credentials

Other properties include **Unconditional Privacy (UP)**, also defined as *ever-lasting privacy*. Unconditional privacy is said to be given when information about a vote is guaranteed to remain a secret for indefinite amounts of time. The goal of UP is that no more information about a vote shall be leaked than the outcome of the vote reveals itself. This property usually relies on the assumption that the asymmetric cryptography used in most systems and remains secure [18].

Vote Verifiability. The verifiability of a vote is a major component in building and ensuring trust in any kind of voting system. In [51], the following forms of verifiability are described:

- **Individual Verifiability (IV)** is given when a voter can verify that his cast vote has reached the intended destination (PSBB) and has not been altered. This could be done in the form of a cryptographic proof. The property on its own is not sufficient to allow for efficient and secure E-Voting as it relies on trusting other voters to verify their cast votes and, therefore, public audits are not possible in an efficient manner [23, 51].
- **Universal Verifiability (UV)** builds on IV by extending the concept such that it allows for anyone (*e.g.*, another voter or a third-party) to verify the outcome of the vote. That is, to verify that the published ballots correspond with the result [35, 37]. This property must be verifiable by anyone as long as they possess the required verification software [51].

A more practical notion often used to assess E-Voting systems is **End-To-End Verifiability (E2E)** that is composed of three components [33, 37, 51]:

- **Cast-as-intended (CAI)**: the ability to verify that the vote has been transmitted to the voting system without being altered.
- **Recorded-as-cast (RAC)**: the ability to verify that the vote has been registered the same way it was cast.

- **Counted-as-recorded (CAR)**: the ability to verify that the vote has been received and was correctly counted in the ballot.

Another form of verifiability mentioned in [37] is referred to as **eligibility verifiability (EV)**. EV states that it must be possible for anyone to verify that all votes registered and counted were cast by eligible voters. Additionally, it must be guaranteed that at most one vote was cast per voter.

Privacy vs. Verifiability. In general, when speaking of a secure E-Voting system, the ultimate goal is to have a system which is verifiable and protects the voter's privacy [33]. These two properties inherently have contradicting goals and are currently an open research topic with different attempts trying to address this issue by using various combinations of cryptographic primitives to preserve the voter's privacy while improving the verifiability of the E-Voting system [33]. Chevallier et al. [18] claims that universal verifiability cannot simultaneously coexist with unconditional privacy or receipt-freeness [18] and, therefore, the goal of E-Voting system design must be to strive for as much privacy as possible while retaining as much verifiability as possible according to Jonker et al. [33].

3.2 Blockchain Enabled E-Voting

Blockchain technology (see section 2.2.4) offers many desirable properties for E-Voting systems and is frequently used as SPBB in research and academia. Recent work [31, 39, 42] relies on a blockchain to secure the voter's ballot and to provide verifiability.

Notable work includes a proposed blockchain-based E-Voting system by Liu et al. [39] that relies on trust between a voter, a central organizer and an inspector. The voter encrypts the vote with the central organizer's public key before the inspector signs it to cast a ballot. This system assumes that both actors can be trusted.
Hardwick et al. [31] on the other hand propose a system with a private Ethereum blockchain that relies on a central certificate authority to authenticate voters, violating the ballot secrecy should the authority become byzantine.
Lastly, a variant of the Open Vote Network (OVN) is proposed in [42] that offers self tallying of the votes. The votes are publicly available once cast and the final tally can be verified by anyone. This brings with it the drawback of fairness, as the last voter can tally the final result before anyone else. Their solution is based on the Ethereum blockchain and is feasible for roughly 40 voters.

This work is partly based on the efforts done in the Provotum project [41]. Provotum is a proof of concept E-Voting system developed at the University of Zurich. It leverages Ethereum smart contracts and homomorphic encryption to provide privacy and verifiability. The system relies on a central server that handles encryption and generates proofs, which could compromise the privacy and trust of the system should this entity fail. Furthermore, the system does not support on-chain verification of the generated proofs.

3.3 Regulations on Electronic Voting in Switzerland

In Switzerland, the first serious tests in the field of E-Voting was conducted in 2001 by the consortium Vote Électronique, initiated by the Federal Council. Since then, multiple projects were initiated such as CHVote [11], the *Guichet Unique* by the Canton of Neuchatel [21] as well as the E-Voting system by the Swiss Post in collaboration with Scytl.

In 2014, the Federal Chancellery Ordinance on Electronic Voting (VEleS) came into effect and defined new requirements regarding the usage of E-Voting systems for certain percentages of eligible voters. The derived properties from these requirements align with the privacy and verifiability properties defined above. These requirements state that if an E-Voting system shall be made available to more than 30% of the eligible voters in Switzerland, it must ensure individual verifiability. Additionally, if the audience is larger than 50%, it also must ensure universal verifiability [23].

In 2019, the Confederation and the cantons ordered an intrusion test on the Swiss Post's E-Voting system during a span of approximately two weeks. IT security experts found a critical gap in the source code that allows the creation of falsified decryption proofs that would still verify without problems [38]. Results in a security report [49] indicate further problems in the documentation of the E-Voting system and conclude that cast-as-intended verifiability may or may not be sound at the time of writing.

These issues were not well received by the public, many voicing concerns about the alleged security of the system. Currently, there exists a movement pushing for a moratorium[1] for E-Voting, that would forbid voting by electronic means for at least five years.

The future will tell if and how the E-Voting landscape in Switzerland will develop.

[1] `https://e-voting-moratorium.ch` (accessed: 15.01.2020)

Chapter 4

Design

In this chapter, a high-level overview of the system design is given in Figure 4.1. The system consists of the following stakeholders: a blockchain, an identity provider, an access provider, different sealers and a voting authority. In the following sections, each stakeholder is described, its responsibilities discussed and the interactions between each other are explained.

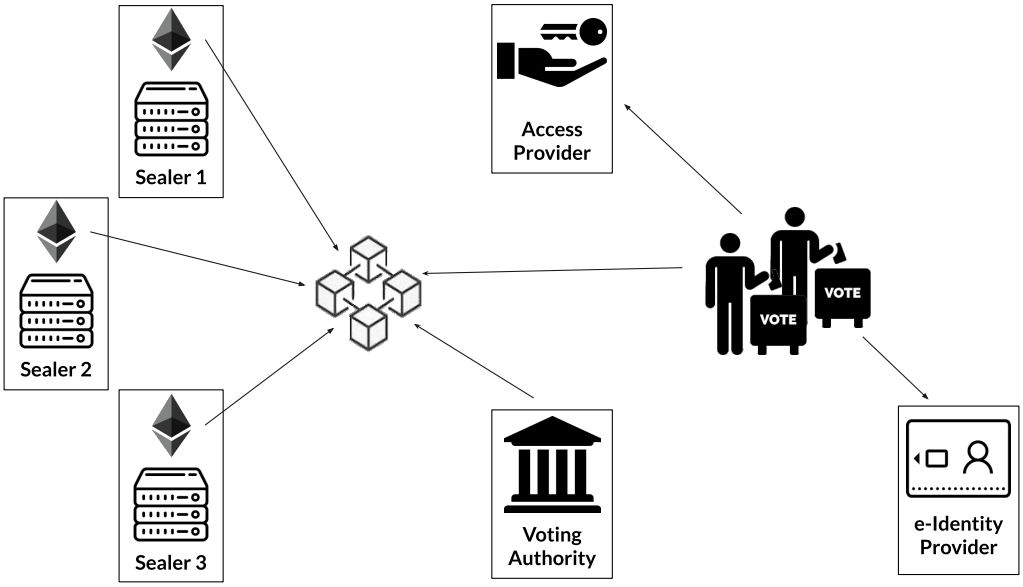

Figure 4.1: E-Voting Architecture & Stakeholders

4.1 Identity- & Access Provider

The identity provider is a trusted third-party responsible for the eligibility verification of potential voters. It provides an authentication service that validates and verifies the electronic identity (E-Identity) of the users trying to gain access to the E-Voting system. After verifying the E-Identity, the identity provider issues a one-time token that will prove the user's authenticity to the access provider.

On the other side, the access provider is a service run by or on behalf of the voting authority. It provides an authorization service that grants only eligible users access to the E-Voting system. The access provider validates the user's one-time token and if successful, funds its blockchain wallet which allows the user to participate in the vote. Together, they provide secure and privacy-preserving authentication and authorization.

4.2 Sealer

The sealer is a service that is required to exist multiple times in the E-Voting system. Its main tasks consist of running a blockchain validator, to participate in the distributed key generation and to tally the votes once the vote has ended. As an example, in a national vote in Switzerland, each canton could take on the responsibility of running a sealer service or at least a few would be required to do so.

4.3 Voting Authority

The voting authority is the administrating body of the vote and acts mainly as a coordinator in the E-Voting system. It is responsible for coordinating the startup of the blockchain with the sealers, creating the voting question, deploying the voting smart contract on the blockchain and opening and closing the vote. In general, the voting authority can be embodied by different organizations (*e.g.*, the municipality, the canton or the national government). The deciding factor is going to be the scope of the vote and the capabilities of the organization.

4.4 Voter

The voter personifies any citizen wanting to participate in the vote. Its only responsibility is to authenticate itself against the identity provider and if deemed eligible, it will be allowed to cast its vote.

4.5 Blockchain

The blockchain is not an active stakeholder with responsibilities as the other stakeholders but a service required to conduct a vote. It provides immutable, append-only storage for all the votes and proofs. New votes can only be appended if the majority agrees. This is necessary to avoid a single point of failure or trust.

Chapter 5

Architecture

This chapter explains the architecture of the voting process, discusses each step in detail and highlights the key points.

5.1 Technology

Since the goal of the project is to establish a working prototype that is straight forward to use and requires only minimal setup and no installation, we decided on a browser-based implementation. Therefore, everything that will be run by the voter needs to be in Javascript. For a better developer experience and a type-safe implementation, Typescript[1] was used. Typescript is a superset of Javascript, meaning all source code will eventually be compiled into Javascript. For the frontend, we use React[2] and MaterialUI[3]. To avoid a lot of context switches, the prototype's backends are also built with Typescript and Node.js[4]. The project uses the Ethereum[5] blockchain as it offers turing-complete smart contracts and has good developer experience. For the Ethereum protocol implementation, the Parity[6] client is used. To ensure fast setup and no installation, all services are containerized using Docker[7] and can be started via a single script. More documentation regarding the implementation can be found in the source code repository.

5.2 Voting Process

In the following section, the voting process is discussed by guiding the reader through an exemplary vote and explaining each step in detail.

[1] `https://www.typescriptlang.org`
[2] `https://reactjs.org`
[3] `https://material-ui.com`
[4] `https://nodejs.org`
[5] `https://ethereum.org`
[6] `https://www.parity.io/ethereum`
[7] `https://www.docker.com`

Figure 5.1: Identity Provisioning. Electoral Register informs the identity provider about the eligible voters.

5.2.1 Identity Provisioning

Before any vote can be carried out, the administrating body of the vote (*e.g.*, the country, canton, or municipality) must decide according to its law and other regulations (*e.g.*, age, citizenship, etc.) who is allowed to participate in the vote. This is necessary since only eligible voters must be allowed to cast their votes. In an electronic vote, this necessarily also needs to be done via an electronic process as otherwise the insecurities and flaws of the traditional process are also introduced into the E-Voting process. The voter verification must be done in such a way that the voter's identity remains secret and cannot be revealed under any circumstances.

Therefore, the identity provisioning in this work's system involves three independent stakeholders: the **identity provider**, the **access provider**, the **voter**. Briefly, the identity provider is an independent trusted third-party that supplies and verifies electronic citizen identifications. The access provider, on the other hand, is a service run by the governing body of the vote (*e.g.*, the canton in case of a cantonal vote or the government in case of a national vote) which authorizes and grants eligible voters access to the system.

As the main focus of this project is to build an E-Voting system using blockchain technology, we simplify some parts of the identity provisioning process and simulate what is necessary for a complete vote to be carried out.

Token Generation

As can be seen in Figure 5.1, as a first step the electoral register of every municipality or canton sends a list of eligible voters identified by a unique ID to the identity provider. Next, the identity provider generates a random one-time token for each eligible voter, shuffles the tokens and sends a list of all tokens to the access provider (Step 2 in Figure 5.1).

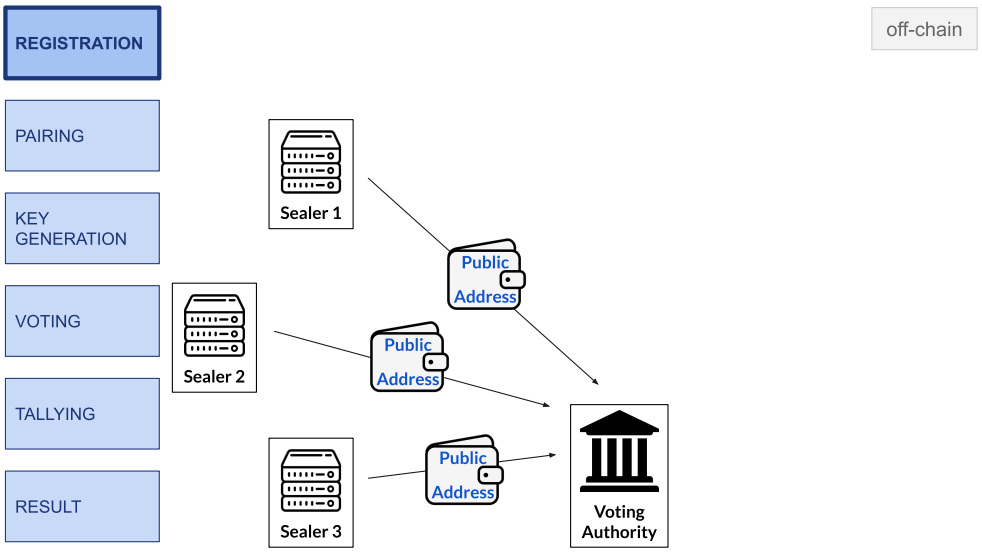

Figure 5.2: Sealer Registration. Each sealer sends the public address of its Ethereum wallet to the Voting Authority.

Identity Provisioning

Later in the process, before a voter can cast his vote he will need to authenticate himself with his E-Identity at the identity provider. The identity provider verifies the E-Identity (*i.e.*, a digital certificate issued by the government containing the unique ID of the voter) and checks if the voter is eligible for this vote. If so, it issues the previously generated one-time token. This is possible since only the identity provider knows the mapping of the voter's unique ID to the generated one-time token.

During the sign-up process, the voter will create an Ethereum wallet and submit his wallet address together with the one-time token to the access provider. The access provider verifies if the one-time token is valid *i.e.*, if it is contained in the list previously received from the identity provider. If so, the access provider stores the address of the voter and marks the one-time token as used. Finally, the access provider will ensure that the voter's wallet has sufficient funds such that he can participate in the vote.

5.2.2 Sealer Registration

This part explains the **REGISTRATION** stage which is the first (actual) stage of the voting process. It consists of the sealer registration. The stage begins with each sealer creating its own Ethereum wallet which consists of a public/private key pair where the public key is considered the public address of the wallet. In a second step, each sealer starts its frontend- and backend service and as can be seen in Figure 5.2, submits the wallet address to the voting authority.

Ethereum wallets can be generated using a tool such as MyCrypto[8]. It is important that

[8] `https://mycrypto.com`

this step must be done offline and before any other action. Also, the private key of the wallet must be kept secret as otherwise someone could impersonate a sealer.

The first stage is finished once all sealers have registered their wallet address with the voting authority. The number of required sealers can be previously decided or a time window can be provided in which everyone willing to participate can register. The voting authority can now advance the state to the next stage.

5.2.3 Blockchain Startup

The second stage called **PAIRING** consists of the sealers starting their Parity clients. The stage begins with all sealers retrieving the finalized blockchain network configuration from the voting authority. The configuration is in JSON format as can be seen in Listing 1 and includes the public addresses of all the sealers. This is required such that all sealers know how to communicate with each other. Additionally, the configuration is used to ensure the wallets of the voting authority, access provider and all sealers have sufficient funds.

```
{
  "name": "VotingPoA",
  "engine": {
    "authorityRound": {
      "params": {
        "stepDuration": "2",
        "validators": {
          "list": [
            "0x98b32c998f16e36cff94b430c656335d682e78dd",
            "0x3e5006ae356a01f2b6fa1c473fb49d73a609d4eb",
            "0x82f876d4bbb1b017e0d730f12d9721e0a2b1fe16"
          ]
        }
      }
    },
    "accounts": {
      "0x004ec07d2329997267ec62b4166639513386f32e": {
        "balance": "1000000000000000000000000"
      },
      "0x004661de90cd1dcb998e8464a0f3c3da9f085950": {
        "balance": "1000000000000000000000000"
      }
    }
  },
}
```

Listing 1: Proof-of-Authority Ethereum Configuration showing validator addresses and pre-funded accounts (shortened)

Now, each sealer will start their Parity client with the retrieved specification and become a validator in the proof-of-authority protocol. Next, the sealer signals to the voting

authority that its client is up and running by attempting to retrieve a URL of another sealer and advertise its existence. This is done by submitting the own URL to the voting authority. If no other sealer has already submitted its URL, this sealer will be considered the boot node. A boot node is a node that others can connect to, to find each other. Any following sealer will use the boot node to connect to the network. Alternatively, the boot node could also be chosen randomly from all connected sealers.

Voting Smart Contract Deployment

Once all sealers have signaled that they are online and connected, the voting authority can proceed by triggering the first action on the blockchain. As can be seen in Figure 5.3, the voting authority creates the voting question and deploys it in a smart contract (*i.e.*, called the ballot contract) to the Ethereum network.

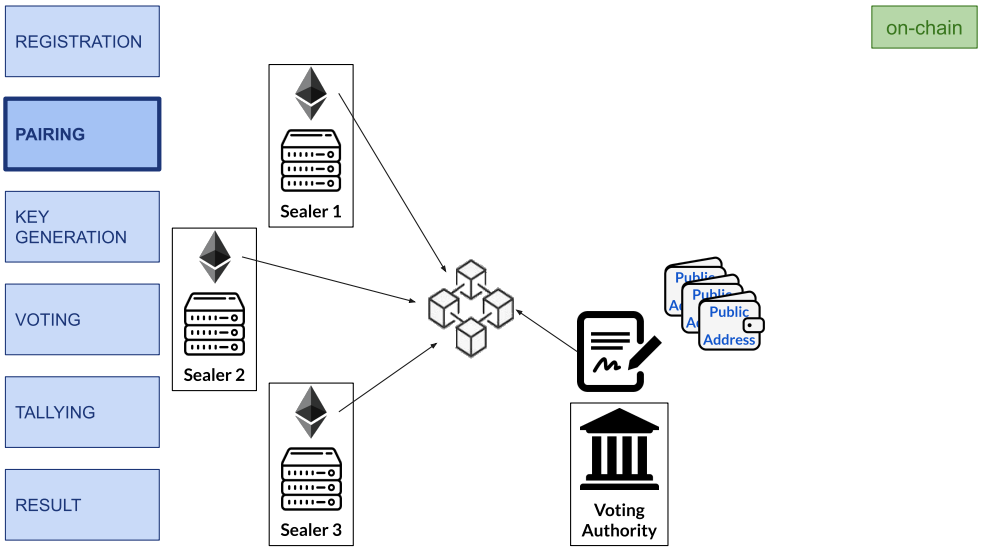

Figure 5.3: Blockchain Startup. Each sealer starts an Ethereum client and the voting authority deploys the ballot smart contract.

The ballot smart contract is responsible for handling and storing all vote specific information (*e.g.*, the vote's system parameters, all encrypted votes, the different proofs, etc.). Also, it includes functionality to verify the different proofs (*i.e.*, for the distributed key generation, the vote membership and the decryption) submitted during the voting process. And, it is responsible for combining the public key shares and the decrypted shares of each sealer to generate the system's public key used for the voting and the vote's result by combining the decrypted shares. After the ballot contract has been successfully deployed, the state advances to the next stage called **KEY GENERATION**.

5.2.4 Distributed Key Generation

In the **KEY GENERATION** stage, all sealers start by generating an ElGamal public/private key pair as shown in Equation 2.1. Next, each sealer generates a proof that

it knows a valid ElGamal private key that belongs to its public key as shown in Equation 2.10. Once the key pair and the proof have been generated, the sealer will directly send both to the ballot smart contract (Step 1 in Figure 5.4). This way trusting any intermediary and, therefore, concentrating the power over the system at a single entity, can be avoided.

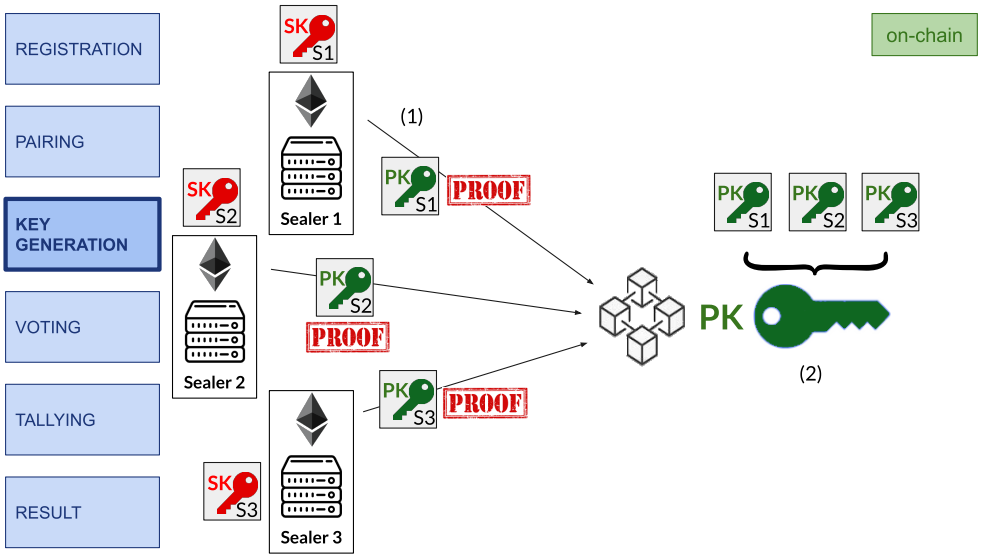

Figure 5.4: Distributed Key Generation. Each Sealer submits its public key and a proof to the blockchain. In a second step, public keys of the sealers are combined into the system's public key.

The ballot contract first verifies the proof as shown in Equation 2.11 and stores the sealer's public key if the proof verification has been successful. Once every sealer has submitted their public key and their proof has been verified, the voting authority can initiate the generation of the system's public key. This is done by triggering a function in the ballot contract. It will compose the system's public key pk_{system} by combining all the sealer's public keys $pk_{sealer_i}, i \in n$, where n is the total number of sealers.

5.2.5 Voting

Once the system's public key has been generated, the voting authority can open the vote. This would most likely be done at a predefined point in time such that the vote can be announced publicly beforehand. By opening the vote, the state is advanced to the **VOTING** stage. This is the only state in which a voter can cast his vote. In any preceding or following state, casting a vote would be prevented by the ballot contract.

Casting Votes

As shown in Figure 5.5, the voting process starts with (1) the voter authenticating himself with his E-Identity at the identity provider. If the eligibility verification is successful, (2)

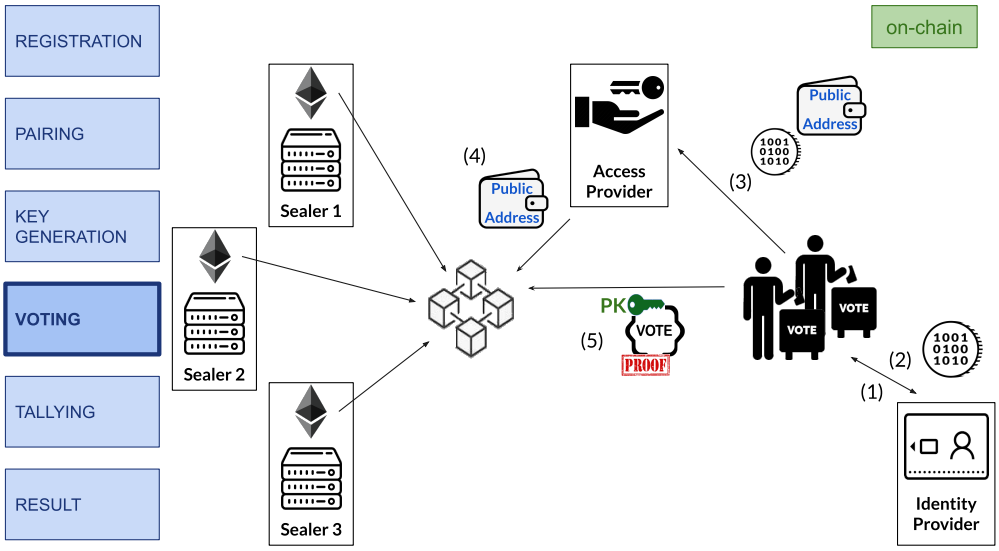

Figure 5.5: Voting. The process of the voter's eligibility verification and system authorization as well as the casting of an encrypted vote including related proof.

the identity provider issues the voter's one-time token. These two steps happen according to the process previously explained in paragraph 5.2.1.

The system (*i.e.*, the service the voter is using in his browser) will generate an Ethereum wallet and use it together with the one-time token to sign up at the access provider (Step 3 in Figure 5.5). The access provider verifies the one-time token and (4) ensures that the voter's wallet has sufficient funds to participate in the vote if the verification was successful.

The voter can now proceed by making a voting decision and encrypt the vote as shown in Equation 2.2. This is done automatically by the system once the voter has selected his choice. In addition to the encrypted vote, the system will generate a proof as shown in Equation 2.14 to show that the encrypted vote contains one of the two possible answers (*i.e.*, yes or no) and nothing else without revealing the actual content of the vote.

Once the vote has been encrypted and the proof has been generated, (5) the system will submit both directly to the ballot contract. The reason why it is submitted directly is again to avoid trusting any intermediary as explained previously in paragraph 5.2.4. The ballot contract first verifies the proof as shown in Equation 2.16 and stores the encrypted vote in the ballot smart contract if the proof verification was successful.

5.2.6 Tallying the Result

Once the voting period has ended which is a previously defined and publicly communicated point in time, the voting authority will close the vote by calling a function of the ballot contract. By that, the system's state is advanced to the **TALLYING** stage.

In this stage, the task of every sealer is to (1) collect all the encrypted votes from the blockchain as shown in Figure 5.6. Next, (2) the sealer will homomorphically add the encrypted votes and create a sum as shown in Equation 2.4 and then decrypt this encrypted sum with his private key share as shown in Equation 2.3. This results in a decrypted share of the vote's final result. The decrypted share by itself is useless and does not convey any information about the outcome of the vote.

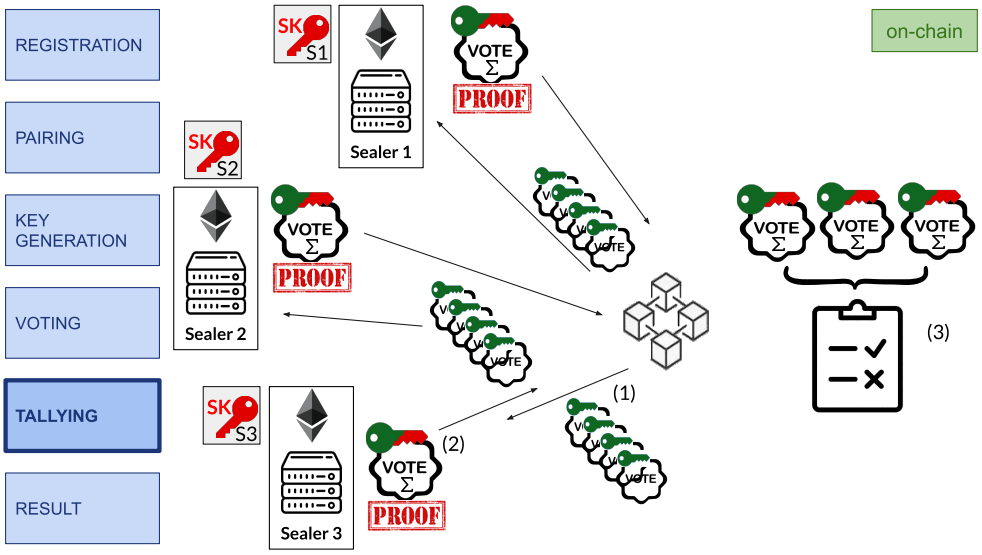

Figure 5.6: Tallying the Result. The process of each sealer computing the encrypted result locally, the decryption, the proof generation and submission, as well as the combination of all decrypted shares, is shown.

To prove the correctness of the sealer's decryption, (2) each sealer will also generate a proof. The proof, as shown in Equation 2.12, proofs that the homomorphic sum has been decrypted using the private key sk_{sealer_i} associated with the public key of the sealer pk_{sealer_i} stored in the ballot contract. This ensures that no one apart from the owner of the private key sk_{sealer_i} (*i.e.*, the sealer) can submit a decrypted share for the respective public key pk_{sealer_i}.

Once both the decrypted share and the proof have been generated, the sealer will send both directly to the ballot contract. Again, this is done for the same reasons as explained in paragraph 5.2.4. The ballot contract verifies the decryption proof as shown in Equation 2.13 and stores the decrypted share if the proof verification was successful.

5.2.7 Result

Once all sealers have submitted their decrypted shares and the ballot contract has published them, the voting authority can trigger the combination of the shares. The ballot contract will combine all decrypted shares d_{sealer_i} and generate the decrypted result of the vote *i.e.*, the number of yes-votes submitted ($result = \sum_{i=0}^{n} d_{sealer_i}$), as shown in Step 3 of Figure 5.6. As the combination of the decrypted shares is done by the smart contract,

which can be read by anyone, there is no need to create a proof for this calculation. The ballot contract automatically subtracts the number of yes-votes from the total number of submitted votes to obtain the number of no-votes: $|votes_{no}| = |votes_{total}| - |votes_{yes}|$. The result is stored in the ballot contract and revealed to everyone once the voting authority advances the state to the final stage **RESULT**.

Chapter 6

Evaluation

In this chapter, the previously shown use case and the system as a whole are evaluated. The proof-of-concept's features and properties are discussed regarding privacy, verifiability, eligibility and security.

6.1 Privacy

The following section evaluates the privacy of the proposed E-Voting system. The E-Voting properties ballot secrecy, receipt-freeness, coercion-resistance, unconditional privacy, and fairness are considered.

6.1.1 Ballot Secrecy

The most fundamental of all privacy properties in an E-Voting system is BS. It requires that the content of a cast vote cannot be revealed under any circumstances and linked to a voter. This system guarantees BS since voters encrypt their votes using the public key of the system that has been composed using each sealer's public key as a share (Equation 2.18). This ensures that no one on their own can decrypt submitted votes. Even if a malicious set of sealers were to collude and attempt to decrypt votes, as long as a single sealer remains truthful it would not be possible for them to decrypt a vote.

Revealing a Voter's Identity

The identity of a voter could potentially be revealed if the IP address of the voter can be pinpointed to an exact location. In less populated areas without skyscrapers or other multi-homed buildings, it might be possible to estimate from the number of people living at the exact location whom the vote might belong to. This is due to the design of the Internet and the way IP addresses are distributed.

6.1.2 Receipt-Freeness

Once the system has generated the encrypted vote (Equation 2.2) and the respective proof (Equation 2.1.5), no information is available regarding the voter's choice. Neither the encrypted vote nor the proof reveals any information about it. On top of that, since every encryption requires a new random parameter r, it is not even possible to prove that two different ciphertexts encrypted the same plaintext. Therefore, RF (*i.e.*, the inability to prove to a third-party that a voter has cast a particular vote) is given in this E-Voting system.

6.1.3 Coercion-Resistance

Although BS and RF are given in this context, CR cannot be guaranteed. If a voter and coercer are at the same physical location, the voter can be forced to abstain or to give up his credentials. Since the voting process is observable in person, the voter can illustrate his voting choice. And, because the voter's wallet address is stored in the ballot contract, he is unable to submit a second vote to overwrite his initial one at a later stage. This is also prevented since multiple submissions are not permitted by Swiss law. Eligible voters are only allowed to cast their vote once [20]. In other countries such as Estonia, voters are allowed to cast their votes multiple times with only the last vote being considered [24].

6.1.4 Unconditional Privacy

UP requires that no assumptions are made regarding the hardness of the cryptographic primitives used to achieve the given level of privacy. However, the ElGamal cryptosystem depends on the hardness of the discrete logarithm problem, as shown in paragraph 2.1.3. As Shor [53] showed, the discrete logarithm problem can be solved in polynomial time using a quantum computer. Therefore, this E-Voting system does not fulfill unconditional privacy.

6.1.5 Fairness

Fairness is achieved if no one can obtain intermediate results of a vote before the end of the voting period. In this E-Voting system, the property is ensured as long as at least a single sealer does not collude. Because only once all sealers have submitted their decrypted share, the share combination can be triggered to generate the result. Since the decryption of the sealer's shares is only initiated once the voting process has officially ended, no intermediate result can ever be revealed.

6.2 Verifiability

The following section evaluates the verifiability of the proposed E-Voting system. The E-Voting properties individual-, universal- and end-to-end verifiability as well as auditiability are considered.

6.2.1 Individual Verifiability

The voter submits the encrypted vote (Equation 2.2) and the respective proof (Equation 2.1.5) directly to the blockchain, as can be seen in Section 5.2.5. Once the smart contract has verified the correctness of the proof (Equation 2.16) and, therefore, the validity of the encrypted vote, the voter will obtain a reference to his vote *i.e.*, the transaction hash of the block including his encrypted vote. As soon as the block containing the transaction has been received by the network and all other sealers agree on incorporating the block, the voter can verify its reference by checking its presence in the transaction log of the blockchain. This fulfills the requirements of the IV property.

6.2.2 Universal Verifiability

The initial requirement to guarantee UV is the public availability of all votes on the SPBB. If a voter or a third-party (*e.g.*, NGO) wants to verify the outcome of the vote, they would have to reproduce and verify the following process.

i All distributed key generation proofs (Equation 2.10) submitted by the sealers need to be verified (Equation 2.11). This guarantees that each sealer knows its associated private key.

ii The creation of the system's public key by the ballot contract needs to be verified. This can be done by manually recombining all public key shares submitted by the sealers.

iii All encrypted votes need to be retrieved from the SPBB and summed (Equation 2.5) to produce the encrypted result. This allows to verify that each sealer has used the same starting point for the decryption *i.e.*, the same encrypted result.

iv All decryption proofs (Equation 2.12) submitted by the sealers need to be verified (Equation 2.13). This guarantees that each sealer has used its private key, previously proven knowledge, to decrypt its share.

v All decrypted shares (Equation 2.20) can be combined to produce the number of yes-votes. This number can then be subtracted from the total number of submitted votes to produce the number of no-votes. Together, they compose the outcome of the vote which can be verified against the published result.

Since anyone can reproduce the summation of the encrypted votes in order to verify the sealers' published input for their decryption, no additional proof is required. On top of

that, anyone can execute the aforementioned process therefore the requirements of this property are fulfilled.

N/N Cooperative Decryption

The big advantage of using distributed key generation is that a single point of failure *i.e.*, an authority with sole access to the system's private key, can be avoided. Even if only a single honest sealer remains, it would still not be possible for a set of malicious sealers to decrypt a vote. The protocol requires that all n parties that have taken part in the key generation have to take part in the cooperative decryption for it to be successful. Unfortunately, the n/n requirement is simultaneously also a risk as sufficiently strong system parameters (*i.e.*, key sizes and prime modulus) would make it impossible to decrypt the result if a single sealer was to lose access to its private key.

Partial Sum Submission

A possible scenario might be that a malicious sealer only sums and decrypts a part of all encrypted votes. Anyone verifying the process would notice that the sealer's published starting point for the decryption (*i.e.*, the encrypted result) is different from the sum that has been generated locally with the votes retrieved from the SPBB.

6.2.3 Auditability

Verifiability is not only important while the voting process is running but even after the voting process has long ended, it should be possible for anyone to verify the outcome of the vote. The approach laid out in paragraph 6.2.2 is possible as long as the sealers maintain the blockchain network and access to it is provided.

6.2.4 End-To-End Verifiability

For an E-Voting system to claim full End-To-End verifiability, it must fulfill the following three properties.

Cast-as-Intended

The E-Voting system allows a voter to encrypt a plaintext vote (Equation 2.2) and generate a proof for its validity (Equation 2.1.5). Both products are directly submitted to the smart contract where the proof is verified (Equation 2.16) and the encrypted vote is stored. This proof can be used to prove the validity of the vote to anyone without any intermediaries. And as only the voter knows the plaintext and the proof generation, the property is fulfilled.

Recorded-as-Cast

Once the smart contract has verified the correctness of the proof (Equation 2.16) and, therefore, the encrypted vote, the voter will obtain a reference to his vote *i.e.*, the transaction hash of the block including his encrypted vote.

The voter can retrieve the proof p and the encrypted vote v at any given time due to the reference (*i.e.*, the transaction hash of the block) obtained after submitting both to the blockchain. By comparing the locally generated proof p_{local} with the obtained proof from the blockchain $p_{blockchain} \stackrel{!}{=} p_{local}$, the voter can ensure the validity of the property. The same holds for the encrypted vote $v_{blockchain} \stackrel{!}{=} v_{local}$.

Counted-as-Recorded

As shown in paragraphs 6.2.2 and 6.2.3, the public verifiability of the result is given by the auditability and universal verifiability. A voter wanting to verify the outcome of the vote can follow the procedure as described in paragraph 6.2.2. By recomputing the sum of all encrypted votes (Equation 2.5) and comparing it to the published starting point for the decryption (*i.e.*, the encrypted result) of each sealer, the incorporation of all submitted votes can be verified.

6.3 Eligibility

The eligibility property ensures that only eligible voters are allowed to participate in the vote. In this proof-of-concept, the identity provider is responsible for the eligibility verification. It does not take part in the voting and is only involved in the verification of the voter's identities. Also, the identity provider is the only one that knows the identity of the voter and the mapping to its one-time token. Since the access provider can only verify the validity and the previous usage of the one-time token, the identity of the voter remains private.

Limitations If the identity provider were to turn rogue, it would be possible for it to create its blockchain wallet and try all generated one-time tokens to gain access to the voting system. This might also go unnoticed as usually not all eligible voters will decide to vote. Also, if the access provider and the identity provider were to collude, it would be possible to link the voter's wallet address to his E-Identity removing the receipt-freeness property.

Implementation The E-Identity verification in this system is not done with digital certificates but instead with a simple username/password-based login. This was done as it does not add any functionality to the proof-of-concept. Further, the assumption is made

that in an actual scenario the requirement of every eligible voter possessing an E-Identity issued by a trustworthy provider (*e.g.*, the local government) is given.

Since the project's main goal is the implementation of the voting process on a blockchain-based E-Voting system and due to the difficulty of the identity provisioning topic, these simplifications and limitations are consciously taken into account.

6.4 Security

Some simplifications regarding production-readiness are made due to time constraints. For example, the communication over secure channels (*e.g.*, the usage of certificates and TLS connections) is not considered. As it does not add any functionality, it was decided against it. It would have been possible to implement but most likely time-consuming and might have made the development process more difficult due to the encrypted traffic.

6.4.1 Side-Channel Attacks

Although the cryptographic library implemented for this work includes timing safe equality checks, other side-channel attacks might still be possible. For example, such as reading directly from specific memory locations during the encryption process might reveal the private key or the random value.

6.4.2 Size of the Prime Modulus P

Due to the limitations of the Ethereum virtual machine only 256-bit integers could be used. This weakens the security of the system as the ElGamal key size can be at most 256-bit. The recommendation nowadays is to use at least 2048-bit for any production setting. Since this is a proof-of-concept, the limitation is not as drastic and can be improved in future work.

6.4.3 Reliability

Our E-Voting system is built on top of a blockchain that ensures redundancy and protects against data loss. Since every sealer stores a complete replica of the whole blockchain on its machine even if some of the sealers malfunctioned, the system could remain operational.

Chapter 7

Summary and Conclusions

While other research focuses on cryptographic building blocks and properties required for E-Voting systems, this work applied a practical approach and investigated the interplay of blockchain technology and electronic voting. Blockchains offer distributed, tamper-proof and append-only data structures which makes them suitable for bulletin boards in E-Voting systems. This work contributes a working and fully documented proof-of-concept implementation of a blockchain-based E-Voting system. While prior work [41] employed a central server architecture, this work's approach proposes a fully distributed system using an Ethereum smart contract as the SPBB. This system offers the following improvements compared to the similar work done in [41]: (i) vote encryption and proof generation is handled on the client's machine instead of a central server; (ii) all proofs are verified on-chain by the smart contract instead of on a central server; lastly, (iii) a distributed key generation scheme adds to the robustness of the system, distributing the power any one node has in the network. The E-Voting system is end-to-end verifiable and offers auditability, ballot-secrecy, and receipt-freeness. Coercion-resistance is not given, as *i.e.*, re-submission of votes is not possible. Currently, these properties hold for integers up to 256 bits (limit of the Ethereum Virtual Machine). This and other shortcomings are addressed in the following section.

7.1 Future Work

Limitations of this work's E-Voting prototype include the identity provisioning scheme and the lack of secure communication channels. Furthermore, other types of votes such as multi-way elections and limited votes [36] could add new functionality to the system. To work towards a secure and fully end-to-end verifiable system, the following subsections suggest possible improvements to the proposed E-Voting system.

7.1.1 Ballot Secrecy Through Onion Routing

As shown in paragraph 6.1.1, it might be possible to reveal a voter's real identity by using the IP address and pinpointing it to an exact location. This problem can be resolved by

incorporating onion routing[1] between the voter and the blockchain network. All traffic would be routed across several hops before entering the network. Additionally, each hop should incorporate a random delay removing the possibility to measure the time taken to pass through various hops.

7.1.2 Blinded Tokens in E-Identity Provision

As was mentioned in Section 6.3, should the access provider and identity provider collude, they would be able to link cast votes to voter identities. One solution to this problem would be a scheme involving blinded voter tokens [15, 29]. In this scheme, the voter would first blind his or her vote and send it to a validator along with authentication details. If the voter is eligible to vote, the validator digitally signs the blinded vote and sends it back. Now the voter can unblind the vote and send it to the tallier-entity of the system. The tallier can now use the signed vote to verify that it was signed by the validator. This confirms that the vote was sent by an eligible voter, but the tallier is unable to find out by whom. In the case of the proposed E-Voting system, the identity provider and access provider would take on the roles of validator and tallier as described.

7.1.3 Multi-Way Elections & Limited Votes

There exist other voting schemes that are not implemented in the current system. While the current system only allows for yes/no votes (or elections with two candidates), multi-way elections and limited votes are not supported. [36]

In **multi-way elections**, voters can select one out of multiple candidates. This can be done by encoding the ballots with a base number N, where N is equal or larger than the total number of eligible voters. Each candidate would then be assigned one power of N, producing ballots N^0, N^1,...,N^{i-1} for i candidates. The numbers are then added to find the winner, *i.e.*, if there are three candidates and the resulting sum is 113, then one candidate received 3 votes whereas the other two only 1. A downside to this approach is that the number of eligible voters needs to be known beforehand. Additionally, large numbers will lead to even more computationally expensive operations.

Limited votes allow one to vote for k out of n total candidates. One approach to implement this would be to look at this as n separate yes/no elections (with 1 candidate). This way, each voter would submit one ballot indicating his choice (yes/no) for every candidate, similar to a vector. For example, if a voter wants to cast a vote for the first and the third candidate but not for the second one, the ballot would look as follows: $(1, 0, 1)$. All vectors would then be added together to find the final result. In the current system, this could potentially be implemented by deploying one smart contract per candidate in an election and combining the overall results at the end. Only the visualization would need some adjustments such that the voter would still have the feeling of taking part in a single election (e.g., only logging in once) even though he is casting multiple ballots for

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

different votes. On the other hand, this would introduce unnecessary overhead and most likely slow the voting process down.

7.1.4 K/N Distributed Key Generation

An alternative algorithm for the decrypted key generation requires k out of n parties not to collude. This way, it is not required that all k parties cooperate, but at least k. In this case, the public key is associated with the knowledge of a polynomial of degree $k-1$. Every authority will know a point on this polynomial and with sufficient k authorities cooperating, the polynomial can be interpolated and exactly determined. This reduces the risk that one party can compromise the system.

7.1.5 Technical Improvements

Apart from the communication over secure channels (*e.g.*, the usage of certificates and TLS connections), the following paragraph highlights other improvements that can be made from a technical standpoint of view.

Blockchain Limitations

The EVM is limited to 256-bit integers, weakening the security of the system as the ElGamal key size can be at most 256-bit. This makes the current system unfit for production use by current standards. For this system to be used reliably and securely on the Ethereum blockchain, a library must be implemented that supports modular arithmetic on integers greater than 2^{256}. Another option lies in the choice of blockchain. This work focused on the Ethereum infrastructure for its maturity and developer experience. Nonetheless, Ethereum is and will always stay a multi-purpose blockchain, which in this case is a large drawback. Other blockchains can be configured more specifically to one's needs, such as Hyperledger Farbic [2] or Parity Substrate [3].

Elliptic Curves over Finite Fields

An alternative approach to larger key sizes would be to use an elliptic curve defined over a finite field due to the ability to use smaller key sizes. For example, a 256-bit elliptic curve public key provides roughly the same security as a 3072-bit ElGamal public key as shown in 2.1.

The ElGamal cryptosystem as defined in 2.1.3 is based on the difficulty of solving the discrete logarithm problem in the finite field F over which the cyclic group Z_p^* is defined. For

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

elliptic-curve-based cryptosystems there exists a similar problem based on the infeasability of finding the discrete logarithm of a random elliptic curve element. The security is given by the ability to compute a point multiplication and the difficulty of finding the multiplicand given the starting and resulting point.

A proof-of-concept for encryption, decryption, proof generation and verification has been implemented in this project's cryptographic library[4]. Due to time constraints the functionality to allow for verification of the proofs on the blockchain has not been implemented.

[4] `http://bcbev.ch/provotum-v2-crypto`

Bibliography

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

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

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

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

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

[6] M. J. Atallah, *Algorithms and theory of computation handbook*. CRC Press, 1999.

[7] J. Benaloh, M. Bernhard, J. A. Halderman, R. L. Rivest, P. Y. A. Ryan, P. B. Stark, V. Teague, P. L. Vora, and D. S. Wallach, "Public evidence from secret ballots," *CoRR*, vol. abs/1707.08619, 2017.

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

[9] T. Bocek and B. Stiller, "Smart Contracts – Blockchains in the Wings," in *Digital Marketplaces Unleashed*. Springer, 2018, pp. 169–184.

[10] D. C. Bochsler, "Remote electronic voting and turnout in the estonian 2007 parliamentary elections," 2010.

[11] Canton of Geneva. E-Voting System. [Online]. Available: https://republique-et-canton-de-geneve.github.io/index-en.html

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

[13] L. Carter and F. Bélanger, "Internet voting and political participation: An empirical comparison of technological and political factors," *Data Base for Advances in Information Systems*, vol. 43, no. 3, pp. 26–46, 2012.

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

[15] ——, "Blind signatures for untraceable payments," in *CRYPTO*. Plenum Press, New York, 1982, pp. 199–203.

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

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

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

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

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

[21] Die Bundesbehörden der Schweizerischen Eidgenossenschaft, "Bericht über die Pilotprojekte zum Vote électronique," *Bundesblatt*, vol. 158, no. 25, pp. 5459–5538, 2006.

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

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

[24] E. Dubuis, "E-Demokratie: E-Voting," in *Handbuch E-Government*. Wiesbaden: Springer Fachmedien Wiesbaden, 2019, pp. 1–14. [Online]. Available: http://link.springer.com/10.1007/978-3-658-21596-5_39-1

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

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

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

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

[29] A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections," in *AUSCRYPT*, ser. Lecture Notes in Computer Science, vol. 718. Springer, 1992, pp. 244–251.

[30] J. Gerlach and U. Gasser, "Three Case Studies from Switzerland: E-Voting," *Berkman Center Research Publication*, p. 17, 2009. [Online]. Available: http://cyber.law.harvard.edu/sites/cyber.law.harvard.edu/files/Gerlach-Gasser_SwissCases_Evoting.pdf

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

[32] M. Hirt and K. Sako, "Efficient Receipt-Free Voting Based on Homomorphic Encryption," in *Efficient Receipt-Free Voting Based on Homomorphic Encryption*, 2000, pp. 539–556. [Online]. Available: http://link.springer.com/10.1007/3-540-45539-6_38

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

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

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

[36] M. Korman, "Secret-Ballot Electronic Voting Procedures Over the Internet," 2007. [Online]. Available: http://www.mkorman.org/ms_thesis.pdf

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

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

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

[40] U. Madise and T. Martens, "E-voting in Estonia 2005. The first practice of country-wide binding Internet voting in the world," *Electronic Voting 2006 - 2nd International Workshop*, no. October, pp. 15–26, 2006.

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

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

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

[44] T. Okamoto, "Receipt-free electronic voting schemes for large scale elections," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1361, pp. 25–35, 1998.

[45] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm," pp. 305–319, 2014.

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

[47] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 228–234, 1980.

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

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

[50] J. Quisquater, M. Quisquater, M. Quisquater, M. Quisquater, L. C. Guillou, M. A. Guillou, G. Guillou, A. Guillou, G. Guillou, S. Guillou, and T. A. Berson, "How to explain zero-knowledge protocols to your children," in *CRYPTO*, ser. Lecture Notes in Computer Science. Springer, 1989, vol. 435, pp. 628–631.

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

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

[53] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999.

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

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

[56] N. Szabo. (1994) Smart Contracts. Last accessed: 12.02.2020. [Online]. Available: https://web.archive.org/web/20160323035617/http://szabo.best.vwh.net/smart.contracts.html

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

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

Abbreviations

BS	Ballot Secrecy
CAI	Cast-as-intended
CAR	Counted-as-recorded
CR	Coercion-Resistance
DKG	Distributed Key Generation
E2E	End-To-End Verifiability
EV	Eligibility Verifiability
EVM	Ethereum Virtual Machine
IV	Individual Verifiability
NGO	Non-Governmental Organization
NIZKP	Non-Interactive Zero-Knowledge Proof of Knowledge
PoA	Proof of Authority
PoW	Proof of Work
RAC	Recorded-as-cast
RF	Receipt-Freeness
SPBB	Secure Public Bulletin Board
TTP	Trusted Third Party
UP	Unconditional Privacy
UV	Universal Verifiability
ZKP	Zero-Knowledge Proof of Knowledge

List of Figures