University of Zurich[UZH]

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

Security Analysis and Improvements of a Blockchain-based Remote Electronic Voting System

Alexander Hofmann
Zürich, Switzerland
Student ID: 11-916-863

Supervisor: Christian Killer, Bruno Rodrigues
Date of Submission: November 18, 2020

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

ifi

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

Abstract

Voting is one of the cornerstones of any democracy. With Electronic Voting (E-Voting), the Swiss confederation wants to advance Switzerland's voting and electoral system to the digital age. One of the main targets of its e-government initiative is the development of a Remote E-Voting (REV) system, with which voters can easily vote from their own devices via the internet. However, this electronic channel, which is proposed as an additional option to in-person and postal voting, poses unique challenges. Digitizing the human right of ballot secrecy to REV systems is not straightforward. Indeed, enabling the verifiability required in order for any voter to check that their vote was counted, is in direct opposition to privacy. Blockchains (BC) promise to bring many of the properties we look for in a voting system: transparency, resilience, tamper-proof, and decentralization. Switzerland, with its federalistic structure, provides a perfect political analogy for implementing a REV system, where trust is distributed among multiple authorities. Canton and municipalities can collaborate to establish a network running the voting software, removing the need to trust any single entity with the correctness of the vote – it would provide transparency to voters and resilience against voting suppression.

This thesis conducts a security analysis of the Provotum REV system in order to assess risks and threats to the current design. Based on the results of the security audit, and improved design for a fully decentralized voting system is proposed. The new architecture brings an advanced notion of voter privacy to BC-based voting: Receipt-Freeness – a measure against vote selling. By introducing a new authority called *Randomizer*, the voter can not prove to a vote buyer how she voted while maintaining the ability to verify the correctness of her vote. Finally, the new scheme is analysed proving that Provotum 3.0 achieves Receipt-Freeness while maintaining Ballot-Secrecy and verifiability. The evaluation includes a scalability analysis, showing any cryptographic material can be generated and verified in linear time, proving that the system can scale to nation-wide elections and votes.

Zusammenfassung

Die Stimmabgabe ist ein wichtiger Eckpfeiler jeder Demokratie. Mit dem elektronischen Stimmkanal (E-Voting) will die Eidgenossenschaft die Schweizer Kultur und Tradition der politischen Rechte in das digitale Zeitalter bringen. Einer der Kernpunkte der schweizerischen E-Government-Initiative ist die Entwicklung eines Systems zur elektronischen Stimmabgabe, mit welchem die Stimmberechtigten bequem von ihren eigenen Geräten aus über das Internet abstimmen können. Dieser elektronische Kanal, der als zusätzliche Option zur persönlichen und postalischen Stimmabgabe vorgeschlagen wird, birgt jedoch einzigartige Herausforderungen. Es ist nicht einfach, das Menschenrecht auf Privatsphäre in Systeme zur elektronischen Stimmabgabe einzubringen. In der Tat steht die Ermöglichung der Verifizierbarkeit, die erforderlich ist, damit jeder Wähler überprüfen kann, ob seine Stimme gezählt wurde, in direktem Widerspruch zur Privatsphäre. Im Laufe der Jahre haben viele Fortschritte in der Kryptographie Kompromisse zwischen Datenschutz und Verifizierbarkeit vorgeschlagen. Ein Beispiel dieses Fortschritts sind Blockchains, welche der Eigenschaften beinhalten, nacwelche für ein Wahlsystem nötig sind: Transparenz, Belastbarkeit, Manipulationssicherheit und Dezentralisierung. Die Schweiz, mit ihrer föderalistischen Struktur, bietet eine perfekte politische Grundlage für die Einführung eines E-Voting Systems, bei welchem das Vertrauen auf mehrere politischen Ebenen verteilt ist. Kanton und Gemeinden können zusammenarbeiten, um ein Netzwerk aufzubauen, das die Abstimmungssoftware betreibt. Ein solches System würde es überflüssig machen, einer einzigen Instanz die Korrektheit der Abstimmung anzuvertrauen. Es würde den Wählern Transparenz und Widerstandsfähigkeit gegen die Unterdrückung der Stimmabgabe bieten.

In dieser Arbeit wird zunächst eine Sicherheitsanalyse des aktuellen Zustands des Provotum-Prototyps durchgeführt, um mögliche Verbesserungsmöglichkeiten aufzuzeigen. Basierend auf den Ergebnissen dieser Prüfung wird ein Entwurf für ein dezentrales Wahlsystem vorgeschlagen. Die neue Architektur bringt einen fortgeschrittenen Begriff des Wählergeheimnisses zur blockchain-basierten Abstimmung: Receipt-Freeness – eine Maßnahme gegen den Verkauf von Stimmen.Durch die Einführung einer neuen Autorität namens *Randomizer* ist der Wähler nicht in der Lage, einem Stimmenkäufer zu beweisen, wie er gewählt hat, während er gleichzeitig die Möglichkeit behält, die Richtigkeit seiner Stimme zu überprüfen. Schließlich wird eine Bewertung der Architektur in Bezug auf die Skalierbarkeit vorgelegt, die zeigt, dass jegliches kryptographisches Material in linearer Zeit generiert und verifiziert werden kann, was beweist, dass das System für landesweite Wahlen und Abstimmungen skalierbar ist.

Acknowledgments

I would like to express my gratitude to those people that have supported me, in many different ways, in order to reach this important milestone. First and foremost, I wish to thank Christian Killer, for his valuable input and the many interesting discussions we've had. He consistently allowed this thesis to be my own work, but steered me in the right direction whenever I needed it. I want to thank Prof. Dr. Burkhard Stiller for the opportunity to work on this fascinating topic within his research group.

Finally, I would like to spend a few words to express my gratitude to those people, who have supported me throughout my entire academic career. To Tramy for her unfailing support and encouragement. And to Elizabeth, for always believing in me.

This accomplishment would not have been possible without all of them.

Contents

Chapter 1

Introduction

"*Enabling equal participation for all and strengthening solidarity*". With these words, the Swiss federal government opens its statement on the critical objectives of its digitization strategy called "Digital Switzerland" [1]. One of its main fields of action is enabling electronic channels for political involvement, which includes Remote Electronic Voting (REV) systems. Following the "security before speed" principle, Switzerland has a long-standing history of testing electronic voting, dating back over 15 years [2]. REV systems pose the unique challenge of allowing citizens to express their votes remotely, from an uncontrolled environment [2]. Legal and technical requirements increase the difficulty in digitizing elections and referenda (e.g., privacy and verifiability properties are in direct opposition to one another). While most people view the introduction of a REV system favorably [3], there is also a minority that sees it as a threat and wants to see it forbidden in Switzerland until it reaches a level of security comparable to that of postal voting [4]. REV presents beneficial properties that are absent in a traditional Remote Postal Voting (RPV) system. For instance, the prevention of invalid votes, the speed of tallying results, various accessibility improvements, convenience and increasing degrees of verifiability [2].

Since its rise to popularity, Blockchains (BC) have been described as the missing key necessary to implement REV in a transparent, resilient, tamper-proof manner, and which requires no trust in a single entity. Indeed, BCs offer many of these properties, empowering voters to vote and verify the voting procedure themselves [5]. However, merely shifting voting over a BC is not enough to secure the process. Hosting an election or referendum introduces new attack vectors and risks which have to be assessed before deploying it on a large scale.

The CSG@Ifi follows various approaches to research on BC-based REV [6]. One of these approaches is the Provotum project [5, 7, 8], which designed and implemented a BC-based REV system formed on a Proof-of-Authority BC as a Public Bulletin Board, which is the central element of any REV system. Provotum 1.0 was first developed in 2018 and resulted in a second version prototyped in 2020 to improve on limitations of the first version. However, the authors of Provotum 2.0 identified limitations that need to be mitigated before such a system can be used in a real-world scenario [9]. For instance, the hard key size limitation on Ethereum's uint256, the Identity Management difficulties seen in the provisioning, and funding of Ethereum accounts allowed to vote, mean that the system

in its current form cannot be employed in any real world setting due to basic security concerns. Potential future work was also identified, such as, possible enhancements of ballot privacy by employing onion routing.

Thus, this work proposes Provotum 3.0, or ProvotumRF, an evolution of Provotum 2.0, which takes the lessons learned over the last 20 years to propose a scheme which achieves Receipt-Freeness on a public permissioned network.

1.1 Description of Work

This thesis's overarching goal is to tackle the limitations faced in the current state of Provotum 2.0. It comprises the possibility to design a new voting protocol and software architectures, requiring the exploration of novel ways to implement such a BC-based REV system. However, to propose a more suitable system or approach, the goal entails a security analysis of Provotum 2.0 to determine risks to the system, which would add to the already known limitations. The newly designed system will be evaluated in terms of privacy and verifiability, as well as usability and scalability. More concretely, this includes evaluating if Provotum 3.0 can be applied in a real-world election scenario, being developed and deployed using modern software and hardware tools, while retaining its properties.

1.2 Thesis Outline

This thesis is structured as follows. Chapter 3 presents the relevant concepts in cryptography and voting system properties needed in order to understand the rest of the thesis. Related work is discussed in Chapter 4. Chapter 5 conducts a security analysis of Provotum 2.0 to enumerate attack vectors and limitations. Following the security analysis, in Chapter 6 improvements on the architecture of Provotum 2.0 are proposed. Chapter 7 describes the implementation details of the Provotum 3.0 prototype. Chapter 8 evaluates the proposed design and prototype in terms of privacy and verifiability, as well as scalability. Finally, Chapter 9 concludes this thesis, discussing this thesis' achievements and possible future work.

Chapter 2

Cryptographic Primitives

Electronic voting systems make used of a wide range of cryptographic tools (primitives). The adoption of these algorithms varies from protecting the secrecy of the vote, to message authentication. This section introduces the most relevant concepts and algorithms needed in order to understand the theoretical arguments of this work.

2.1 Hash functions

A hash function accepts a variable-size message M and produces a fixed sized output, called a digest or also fingerprint [10]. A secure hash function H must possess the following properties:

- H can be applied to block of data of any size.
- H produces fixed sized digests.
- $H(x)$ must be efficient to compute.
- *Pre-image resistance:* for any $h = H(x)$, it must be infeasible to find x.
- *Second pre-image resistance or weak collision resistance:* for any x, it must be infeasible to find y, where $y \neq x$, s.t. $H(x) = H(y)$.
- *(Strong) collision resistance:* it must be infeasible to find any pair (x, y), s.t. $H(x) = H(y)$.

In general, cryptographic hash functions must be efficient to compute for a given message, but inverting them should not, which gives them their "one-way" property. Such a secure cryptographic hash function can be employed as a primitive for a variety of applications, *e.g.,* message authentication codes, digital signatures, or as a replacement for a random oracle in non-interactive proofs. Common implementations to electronic voting are commitments schemes, privacy-preserving electronic signature protocols, and non-interactive ballot validity proofs.

2.2 Public-Key Cryptography

Public-key cryptography is asymmetric, meaning that distinct keys are used for encryption and decryption of messages, or for signing and verifying digital signatures [10]. While traditional shared-key cryptography mostly relies on manipulation of bit patterns, public-key cryptography relies on mathematical functions and the intractability of certain classes of problems, such as the discrete logarithm or the prime factorization problem [10]. A public-key scheme consists of the following components: *(i)* a plaintext m, *(ii)* an encryption algorithm E, *(iii)* a public/private key pair (pk, sk), *(iv)* an encrypted ciphertext c, and *(v)* a decryption algorithm D. It should be infeasible to derive the private key from the public key. The subsequent relation ties the components together:

$$m = D(c, sk) = D(E(m, pk), sk) \tag{2.1}$$

Generally, public-key crypto systems require more computations than their symmetric counterparts [10]. For this reason, public-key cryptography is usually applied in scenarios were the messages to encrypt (or sign) tend to be relatively small, such as digital signatures and key exchanges. Examples are RSA [11], Diffie-Hellmann Key Exchange [12], and the ElGamal crypto system [13], described in Section 2.5.

2.2.1 Digital Signatures

NIST defines a digital signature as "*The result of a cryptographic transformation of data that [...] provides a mechanism for verifying origin authentication, data integrity and signatory non-repudiation*" [14]. Digital signatures are cryptographic schemes giving a means to demonstrate that a message comes from a specific known sender, and that the message was not manipulated along the way. Whereas encryption schemes provide protection against eavesdropping, signatures deliver authentication and integrity.

Definition 2.2.1 (Digital Signature Scheme) *A digital signature scheme is a triple of algorithms (`KeyGen, Sign, Verify`), where sk is the signing key, and vk is the verification key [15], s.t.*

$$\begin{aligned} (sk, vk) &= \texttt{KeyGen()} \\ \sigma &= \texttt{Sign}(sk, m) \\ b &= \texttt{Verify}(vk, m, \sigma) \end{aligned} \tag{2.2}$$

A signature is valid if $b = 1$.

A concrete instance of this algorithm is given by the RSA signature scheme [11].

`KeyGen:` choose two random, distinct prime numbers $p,\ q$ and compute

$$\begin{aligned} N &= pq \\ \phi(N) &= (p-1)(q-1) \end{aligned} \tag{2.3}$$

select e, s.t. e is coprime to $\phi(N)$ and $e < \phi(N)$

$$\begin{aligned} d &= e^{-1} \pmod{\phi(N)} \\ (sk, vk) &= ((N,d),\ (N,e)) \end{aligned}$$

`Sign:` given any hash function H, compute the signature σ of the message m

$$\sigma = H(m)^d \pmod{N} \tag{2.4}$$

`Verify:` verify the signature by computing

$$\sigma^e \stackrel{!}{=} H(m) \pmod{N} \tag{2.5}$$

2.2.2 Decisional Diffie-Hellmann Assumption

The decisional Diffie-Hellmann Assumption is an assumption on the computational intractability about solving a discrete logarithm in a cyclic group.

Definition 2.2.2 (Decisional Diffie-Hellmann Assumption) *Given a cyclic group G of prime order q and generator g, the DDH assumption states that for any triple $(a,b,c) \in_R Z_q$, the following triple cannot be distinguished by a polynomially bounded Turing machine: $(g^a, g^b, b^c), (g^a, g^b, g^{ab})$, with bound $n = log(q)$. Essentially, this means that the value $g^{ab} \in_R G_q$ cannot be distinguished from any other $c \in_R G_q$ [16].*

2.3 Blind Signatures

A blind signature is a cryptographic scheme based on public cryptography, first proposed by Chaum [17]. It enables a user to request a digital signature of a message, while keeping its content secret to the signer. In practice, this can be employed to ensure that a signer cannot trace the usage of its signature. The common analogy adopted to describe this process is that of using an envelope, containing a message written on carbon paper which needs to be signed. The user seals the envelope and gives it to a notary for signing, transferring the signature from the envelope to the carbon paper. The user can afterwards open the envelope, revealing the signed message.

Blind signatures have often been proposed in literature as a mechanism to prove to voting systems the legitimacy of a voter casting a ballot. In most schemes proposed, the voter prepares a ballot, blinds it, authenticates to the authority, and sends the blind ballot. The authority signs it and returns it to the voter. The voter can now unblind the payload, revealing the signature. The voter then casts the ballot in plain text with the signature, which the receiving party can verify as a normal digital signature.

Definition 2.3.1 (RSA Blind Signature) *An RSA blind signature is defined as a tuple of algorithms (`KeyGen, Blind, Sign, Unblind, Verify`) [17].*

`KeyGen:` obtain (sk, vk), as described in Equation 2.3

$$(sk, vk) = ((N, d),\ (N, e)) \tag{2.6}$$

`Blind:` choose a random, blinding factor $r \in_R \mathbb{Z}_N$, s.t. r is coprime to N

$$(b, r) = (H(m) \cdot r^e,\ r) \pmod{N},\ r \in_R \mathbb{Z}_N^* \tag{2.7}$$

`Sign:` compute blind signature σ', using the standard RSA signature (Equation 2.4)

$$\sigma' = b^d \pmod{N} \tag{2.8}$$

`Unblind:` unblind it to reveal the signature σ

$$\sigma = \frac{\sigma'}{r} \pmod{N} \tag{2.9}$$

`Verify:` verify using the standard RSA signature verification (Equation 2.5)

$$\sigma^e \stackrel{!}{=} H(m) \pmod{N} \tag{2.10}$$

2.4 Homomorphic encryption

Homomorphic encryption schemes are a form of encryption, with the property of being able to perform operations on a ciphertext, without the need of decrypting it first.

In electronic voting, this is often used in order to sum up votes for a candidate, without decrypting the individual votes, providing privacy to voters.

This can be formulated as

$$\forall\ m_1, m_2 \in M,\ E[m_1 \oplus m_2] = E[m_1] \otimes E[m_2], \quad M \text{ message space} \tag{2.11}$$

Homomorphic schemes are categorized based on the degree of homomorphism they submit: *i.e.,* a single, specific operation, or arbitrary operations. Examples of known schemes which provide partial homomorphism under a single operation are ElGamal [13] (widely used in electronic voting), RSA [11], and the Pailler crypto system [18]. This thesis, follows in the step of many other REV schemes proposed in literature (such as [19, 20, 21]), using ElGamal.

2.5 ElGamal Cryptosystem

ElGamal is an asymmetric encryption scheme, proposed by Taher ElGamal [13]. It draws its security on the basis of the DDH assumption (Definition 2.2.2): the hardness of computing discrete logarithms.

Definition 2.5.1 (Diffie-Hellmann group) *A Diffie-Hellmann group is a group $G \subset \mathbb{Z}_p^*$ of prime order q, generator g, with p, q primes and $(p-1)/2 = q$. The values (p, q, g) are public [15].*

These groups have been standardized in RFC [22] and are readily available for key generation.

Definition 2.5.2 (ElGamal Encryption) *The ElGamal scheme defines a tuple of algorithms (`KeyGen`, `Encrypt`, `Decrypt`), where the message- and ciphertext-space are defined over a Diffie-Hellmann group* G *of prime order q and generator g [13].*

`KeyGen:` choose or obtain parameters (p, q, g). Pick sk at random from $\mathbb{Z}_q$ and compute:

$$pk = g^{sk} \pmod{p} \tag{2.12}$$

where pk is the public-key and sk the private-key.

`Encrypt:` choose a random nonce $r \in \mathbb{Z}_q$. Encrypt message $m \in \mathbb{Z}_p$ by computing

$$E[m] = (c, d) = (g^r,\ m \cdot pk^r) \pmod{p} \tag{2.13}$$

where the pair (c, d) is the ciphertext.

`Decrypt:` compute

$$m = \frac{d}{c^{sk}} \pmod{p} \tag{2.14}$$

Multiplicative Homomorphism The scheme, as defined in Equation 2.11, is homomorphic over $(G,\ \cdot)$. For a simple proof, given two messages $m_1,\ m_2 \in G$:

$$\begin{aligned} E[m_1] \cdot E[m_2] &= (g^{r_1},\ m_1 \cdot pk^{r_1}) \cdot (g^{r_2},\ m_2 \cdot pk^{r_2}) \\ &= (g^{r_1+r_2},\ m_1 \cdot m_2 \cdot pk^{r_1+r_2}) \\ &= (g^{r},\ (m_1 \cdot m_2) \cdot pk^{r}), \quad r = r_1 + r_2 \\ &= E[m_1 \cdot m_2] \end{aligned} \tag{2.15}$$

In order to turn this scheme into something useful for the purpose of counting votes, we need it to be homomorphic over $(G, +)$. We replace the `Encrypt` (Equation 2.13) and `Decrypt` (Equation 2.14) functions as follows:

$$(c, d) = (g^r,\ g^m \cdot pk^r) \pmod p \tag{2.16}$$

and

$$g^m = \frac{d}{c^{sk}} \pmod p \tag{2.17}$$

Determining m from Equation 2.17 involves solving a discrete logarithm. However, due to the fact that the message space M in e-voting is usually small and bound to the amount of voters, this can be done quickly by naively bruteforcing it with a runtime of $\mathcal{O}(n)$. Alternatively, an algorithm such as Baby-step giant-step [23] can be used, which has a runtime of $\mathcal{O}(\sqrt{n})$.

Additive Homomorphism Showing that the scheme is now additively homomorphic is simple:

$$\begin{aligned} E[m_1] + E[m_2] &= (g^{r_1},\ g^{m_1} \cdot pk^{r_1}) \cdot (g^{r_2},\ g^{m_2} \cdot pk^{r_2}) \\ &= (g^{r_1+r_2},\ g^{m_1} \cdot g^{m_2} \cdot pk^{r_1+r_2}) \\ &= (g^{r},\ g^{m_1+m_2} \cdot pk^{r}), \quad r = r_1 + r_2 \\ &= E[m_1 + m_2] \end{aligned} \tag{2.18}$$

With this construction, we can also define the multiplication of a scalar k with a ciphertext

$$\begin{aligned} k \otimes E[m] = E[m]^k &= (g^{r},\ g^{m} \cdot pk^{r})^k \\ &= (g^{rk},\ g^{mk} \cdot pk^{rk}) \\ &= (g^{r'},\ g^{mk} \cdot pk^{r'}), \quad r' = rk \\ &= E[km] \end{aligned} \tag{2.19}$$

Security Properties To any observer, including a voter who encrypts a chosen plaintext m, the scheme is IND-CPA, that is indistinguishable in a chosen-plaintext attack scenario [15], considering that the message m is multiplied with a uniformly random value pk^r. Thus, two ciphertexts cannot be correlated, even if the original plaintext messages are the same. Additionally, given the homomorphic property, the scheme is by construction malleable.

2.6 Distributed Key Generation and Cooperative Decryption

Distributed Key Generation (DKG) and Threshold encryption are techniques that can be used to allow multiple authorities to participate in the creation of a key pair [24, 25]. This opens up the possibility of having public crypto systems, without the necessity to centralize the ability of decrypting a message to a single entity. Instead, a threshold of authorities are necessary in order to decrypt a message encrypted with the public-key. While threshold encryption allows $t \leq N$ authorities to collaboratively decrypt a message, in the following DKG scheme for ElGamal all N authorities will be needed, *i.e.,* $t \stackrel{!}{=} N$. Such a scheme is employed in both Provotum 2.0 [7], as well as in the system presented in this thesis.

Distributed Key Generation Before the key generation can start, the N authorities need to agree on the encryption parameters (p, g, q). This step can be done publicly. Every authority creates an ElGamal key pair $(pk_i, sk_i),\ i \in \{1, N\}$, as described in Definition 2.5.2. The public-keys are published in order to create the common public-key.

$$pk = \prod_{i=1}^{N} pk_i \pmod{p} \tag{2.20}$$

Encryption Any party which wishes to encrypt a message can do so, following the ElGamal scheme as presented in Definition 2.5.2 using the key computed in Equation 2.20.

Decryption Decryption requires all N authorities to collaborate. First, each authority decrypts the ciphertext $e = (c, d) = E[m, pk]$, using the secret-key sk_i, producing a decrypted share d_i. This is achieved by applying a different algorithm as the one discussed previously.

$$d_i = c^{sk_i} \pmod{p} \tag{2.21}$$

The decrypted share alone does not yet reveal the plaintext, nor does it convey any information useful for cryptanalysis. Afterwards, all N decrypted shares d_i are combined to reveal the plaintext m.

$$m = c_2 \cdot (\prod_{i=1}^{N} d_i)^{-1} \pmod{p} \tag{2.22}$$

Due to the malleability of the ElGamal scheme, it is possible for a malicious authority to return undetected any value as its decrypted share d_i. For this reason, a zero-knowledge proof is usually attached to prove that a share was decrypted honestly, without having to reveal the authority's secret-key sk_i (Section 2.7).

2.7 Zero-Knowledge Proofs

Zero-knowledge proofs (ZKP) are a technique with which a prover can demonstrate to a verifier knowledge about a secret, without having to leak the secret itself [26, 27].

A ZKP must satisfy the following three properties [26]:

1. **Completeness:** if the statement is true, the honest verifier will accept it.
2. **Soundness:** if the statement is false, it is not possible to convince an honest verifier that the statement is true.
3. **Zero-knowledge:** no malicious verifier can learn anything about the secret from the proof, except for its correctness.

It is important to note, that a ZKP provides probabilistic guarantees about its soundness: *i.e.,* the probability that a malicious prover can craft a proof for a false statement that will be accepted by an honest verifier, is negligible, but non-zero. In E-Voting systems, ZKP can be used to determine that a ballot is valid without leaking the contents of a vote, or to test the correctness of a decryption without revealing the private-key used.

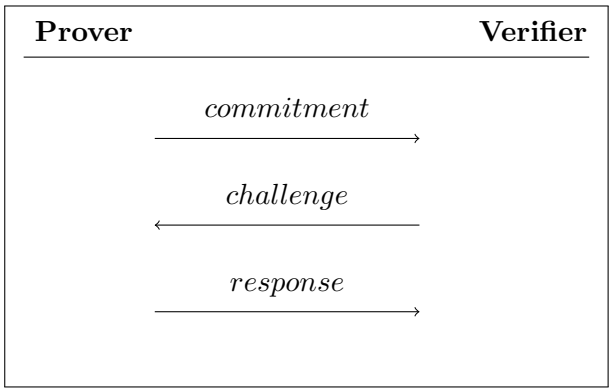


Figure 2.1: Σ proof structure

There are essentially two types of ZKP: *interactive* and *non-interactive* zero-knowledge proofs (NIZKP). *Interactive proofs* are usually formulated as a challenge-response game, in which the prover commits to a statement, the verifier sends a challenge to the prover, and the prover responds (Figure 2.1). The commitments, together with the response, can be used to verify the statement to prove. Such three-move protocols are called Σ proofs [26, 27]. *Non-interactive proofs* do not require interaction and can be generated by the prover alone. In the same way, the verifier can verify the proofs without requiring the voter. This makes them ideal for generating and publicly publishing in a fire-and-forget way. One common technique to turn an interactive protocol into a non-interactive one is via the Fiat-Shamir Heuristic.

2.7.1 Fiat-Shamir Heuristic

The Fiat-Shamir Heuristic [28] is a technique to transform an interactive proof of knowledge into a non-interactive one, by replacing the random verifier's challenge with a hash of the algorithm's transcript, which serves as the random oracle – the source of uniform randomness. Non-interactive zero-knowledge proofs (NIZKP) can then easily be published, allowing anyone to verify a statement, without information leakage.

Different protocols have made use of the Fiat-Shamir Heuristic to reduce communication overhead in remote voting protocols. Bernhard *et al.* [29] identified that two distinct applications of such heuristic exist in literature: a strong variant (sFS) and a weak one (wFS). The strong variant hashes the commitments as well as the statement, while the weak one only hashes the commitments. This can lead to unsound proofs in the case of malicious provers: *i.e.,* proofs that do not actually prove the statement, they are meant to prove.

2.7.2 Schnorr Proof

A Schnorr Proof is a ZKP that can be used to prove knowledge of a private-key sk for a specific public-key pk.

Definition 2.7.1 (Schnorr Proof) *A Schnorr Proof [30] is a proof of knowledge of a secret value x, such $X = g^x \ (mod\ p)$, where (g, p, X) are public values.*

For a key pair (sk, pk) and public parameters p, g, q, we compute a proof of knowledge of the secret sk using the formula in Figure 2.2.

Prover		**Verifier**
knows $sk,\ pk,\ g,\ p,\ q$		knows $pk,\ g,\ p,\ q$
$a \in_R \mathbb{Z}_q$		
$b = g^a \pmod p$		
$c = H(pk, b)$		
$d = a + c \cdot sk \pmod q$		
	$\xrightarrow{(c,d)}$	compute
		$b = g^d / pk^c \pmod p$
		$c' = hash(pk, b)$
		verify
		$c \stackrel{!}{=} c'$
		$g^d \stackrel{!}{=} b \cdot pk^{c\prime} \pmod p$

Figure 2.2: Schnorr non-interactive proof

2.7.3 Chaum-Pedersen Proof

A Chaum-Pedersen Proof can be used to prove the equality of two discrete logarithm and can be used to prove the decryption of a ciphertext e with a private-key sk, congruent to the public-key pk.

For a given ciphertext $e = (e_1, e_2)$ encrypted using the public-key pk and sk, p, g, q, we compute a Chaum-Pedersen Proof as shown in Figure 2.3.

Prover		**Verifier**
knows sk		
$r \in_R \mathbb{Z}_q$		
$u = c^r \pmod p$		
$v = g^r \pmod p$		
$C = H(pk, e_1, e_2, u, v)$		
$s = r + C \cdot sk \pmod q$		
$d = e_1^{sk} \pmod p$		
	$\xrightarrow{d, u, v, s}$	$C = H(pk, e_1, e_2, u, v)$
		verify
		$c^s \stackrel{!}{=} u \cdot d^C \pmod p$
		$g^s \stackrel{!}{=} v \cdot pk^C \pmod p$

Figure 2.3: Chaum-Pedersen proof

2.7.4 Disjunctive Chaum-Pedersen Proof

A disjunctive Chaum-Pedersen Proof (DCP) is used to prove the validity of an encrypted ballot. Due to the indistinguishability property of ElGamal, once a ballot e has been encrypted, no one can tell whether e encrypts a 0, 1, or even a -1. Thus, we apply the disjunctive Chaum-Pedersen ZKP to prove that the ballot encrypts a value $v \in \{0, 1\}$. We do this by proving that the voter has either encrypted a 0, OR a 1, using two Chaum-Pedersen proofs. The protocol in Figure 2.4 presents the non-interactive proof generation and verification protocol.

2.7.5 Designated-Verifier Proofs

A designated-verifier proof, is a type of knowledge proof which can be verified by a specific (designated) entity, but when transferred to third party, it makes no sense [31]. The proof can be generated using an OR-combination of the statement we wish to prove, and the proof of knowledge of the private-key of the designed verifier. As only the verifier knows the private-key, the verifier knows the statement to be true. Intuitively, this can be

Voter		**Verifier**
knows $v \in \{0,1\},\ \alpha \in_R \mathbb{Z}_q$		knows $e = (c, d)$
s.t. $e = E(v, \alpha) = (c, d)$		
$\lambda\ = 1 - v$		
$c_\lambda,\ r_\lambda,\ r'_v \in_R \mathbb{Z}_q$		
$a_\lambda = g^{r_\lambda}/c^{c_\lambda} \pmod p$		
$b_\lambda = pk^{r_\lambda}/(d/g^\lambda)^{c_\lambda} \pmod p$		
$a_v = g^{r'_v} \pmod p$		
$b_v = pk^{r'_v} \pmod p$		
$C = H(pk, a_0, a_1, b_0, b_1, e)$		
$c_v = C + c_\lambda \pmod q$		
$r_v = r'_v + c_v \cdot \alpha \pmod p$		
$\pi\ = (a_0, a_1, b_0, b_1, c_0, c_1, r_0, r_1)$	$\xrightarrow{\pi}$	for $i \in \{0,1\}$
		$g^{r_i} \stackrel{!}{=} a_i \cdot c^{c_i} \pmod p$
		$h^{r_i} \stackrel{!}{=} b_i \cdot (d/g^i)^{c_i} \pmod p$
		$C\ = H(pk, a_0, a_1, b_0, b_1, e)$
		$c_0 + c_1 \stackrel{!}{=} C \pmod q$

Figure 2.4: Disjunctive Chaum-Pedersen proof

formulated as "*Instead of proving* Θ*, Alice will prove the statement "Either* Θ *is true, or I am Bob*"" [31]. If Bob receives this proof, it will be obvious to him that the statement Θ must be true (or that his key was compromised). However, if he sends the proof to Charlie, Charlie will not be able to distinguish whether the statement is true, or if Bob is simply proving to be himself. The application of designated-verifier proofs for Provotom is explained in Section 6.2.4.

2.7.6 Divertible Proofs

Similarly to blind signatures, divertible proofs can be used to generate new proofs using existing ones, in such a way that the new one cannot be distinguished by any other proof – not even the original one [32].

In divertible proof schemes, an intermediary party is introduced between prover and verifier – posing as the verifier to the prover, and as the prover to the verifier – as a sort of man in the middle (MITM). In the context of this work and that of [33, 34], it is used in order to allow an authority to generate a new DCP for a ballot, which was modified without leaking the vote or the nonce used for the encryption.

In the original paper [32], entire classes of divertible proofs are presented. For this work, we will define a divertible interactive DCP algorithm in Section 6.2.4.

Chapter 3

Background

This chapter focuses on theoretical concepts central to REV systems. In the first part, the main security properties of e-voting systems are discussed. In the second part, blockchains are introduced and it is shown how they can be applied to the context of e-voting systems.

3.1 Remote Electronic Voting Properties

Various desirable properties for voting systems have been defined in literature over the years, as can be seen in Figure 3.1. Different authors tend to describe these in various terms. For this reason, this section aims to give an overview of the terms as used in the context of this work. Jonker *et al.* [35] provide a survey of desirable properties of REV voting systems, which forms the basis of this section. These can be categorized into two groups: privacy and verifiability.

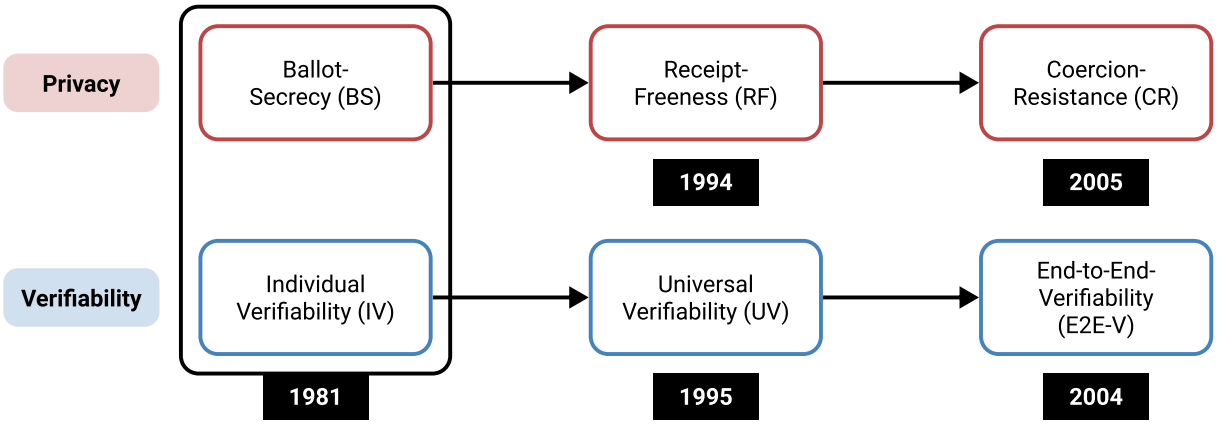


Figure 3.1: Overview and evolution of privacy and verifiability notions in voting systems [35].

Privacy Vote privacy is considered a fundamental human right and is stated at article 21 of the Universal Declaration of Human Rights [36]. Intuitively, removing ballot privacy from the voting process allows adversaries to easily coerce and/or bribe voters. Thus, privacy is a key factor in allowing a democratic voting process. Privacy in this context is often split into *(i)* Ballot-Secrecy, *(ii)* Receipt-Freeness, and *(iii)* Coercion-Resistance.

Ballot-Secrecy can be further classified in subcategories based on the degrees of privacy and assumptions made. However, this is not handled in this work, but we consider the following definition of Ballot-Secrecy.

Definition 3.1.1 (Ballot-Secrecy (BS)) *"A voting system is private if it (and the procedures/process for using it) does not make available additional information on an individual voter's ballot choice(s), beyond that contained in the tally [37]" OR "A voter's vote is not revealed to anyone" [38].*

Further refinements of this definition can be found in literature based on the trust model or the computational power of an adversary [39, 40, 41]. Sometimes, the term *Ballot Privacy* is used interchangeably as a synonym. For this thesis, only BS is used to better distinguish it from other privacy notions. Interestingly, all early works in the field devised systems to produce receipts, allowing voters to prove to others how they voted. Intuitively, the ability to produce a proof of how a user voted can easily be used for coercion and vote buying [42]. This brought about the definition of Receipt-Freeness.

Definition 3.1.2 (Receipt-Freeness (RF)) *"A voter cannot gain information which can be used to prove, to a vote-buyer, how she*[1] *voted" [42].*

Definition 3.1.3 (Coercion-Resistance (CR)) *"A voter cannot interact with a coercer to gain information, which can be used to prove how she voted" [35].*

The core difference between Receipt-Freeness and Coercion-Resistance lies in the type of voter. In the case of Receipt-Freeness, we consider a malicious voter who wishes to sell her vote. For Coercion-Resistance instead it does not matter whether the voter is honest or not. She is coerced by some entity into divulging sensitive information, such as her vote, her credentials to vote and so on. As such Coercion-Resistance is a stronger privacy guarantee than Receipt-Freeness.

Definition 3.1.4 (Unconditional Privacy (UP)) *No information is leaked other than what is leaked by the tally, regardless of computational or time assumptions [39].*

Many voting schemes based on asymmetric encryption (*e.g.,* mix-net and homomorphic encryption) rely on the intractability of mathematical problems, such as the Decisional Diffie-Hellman assumption (DDH) [16]. As such, they do not offer unconditional privacy.

[1] In keeping in line with other works in the field, the voter is female (Alice) and is thus referred to with the feminine pronouns.

Verifiability Once all votes have been cast, the system must be able to prove that the result was tallied correctly, and that the single ballots contain the actual voters' choices. We call this property *Verifiability*. Jonker *et al.* [35] present various definitions of verifiability.

Definition 3.1.5 (Individual Verifiability (IV)) *A voter can verify that the ballot she cast is in the published set of "all" votes and it is unaltered [35].*

Definition 3.1.6 (Universal Verifiability (UV)) *Anyone can verify that the final tally corresponds with the published set of "all" votes [35].*

Definition 3.1.7 (Eligibility Verifiability) *Anyone can verify that any vote in the set of "all" votes belongs to an eligible voter, and each eligible voter voted only once [43].*

Definition 3.1.8 (End-to-end Verifiability (E2E-V)) *End-to-end Verifiability is fulfilled if a voter can verify three properties [35]:*

- *Cast-as-Intended: "her choice was correctly denoted on the ballot by the system".*
- *Recorded-as-Cast: "her ballot was received the way she cast it".*
- *Counted-as-Recorded: "her ballot counts as received".*

Privacy vs Verifiability Intuitively, voter privacy and verifiability are in direct opposition to one another. Indeed, Chevallier *et al.* [44] have shown that a voting system cannot achieve Unconditional Privacy and Universal Verifiability at the same time. They also showed that Universal Verifiability and Receipt-Freeness also cannot coexist if the transcript of the vote depends only on public values and values known to the voter. For this reason, REV Systems should strive to achieve a trade-off between privacy and verifiability.

Definition 3.1.9 (Software Independence) *A voting system is software-independent if an undetected change or error in its software cannot cause an undetectable change or error in an election outcome [45].*

REVs are complex system. Small errors or manipulations can lead to unpredictable errors, which might be exploited by adversaries. As such Software Independence follows the approach of "Verify the election results, not the voting system" [45].

3.2 Blockchain and Public Bulletin Boards

A blockchain (BC) is an append-only, immutable, distributed database, managed by a peer-to-peer network. The state is composed out of blocks of transactions which are cryptographically linked together in a chain. A block is simply a data structure containing a set of transactions and a reference to the previous block in the chain. In this sense, a BC is essentially a reverse linked-list. In order to decide which blocks to include in the chain, a BC defines a consensus mechanism through which the nodes of the p2p network agree on the state of the database. Generally, all peers in the network agree on an initial state of the system, called the genesis block. From there, each new appended block is validated and agreed upon by the network through its consensus algorithm. Some examples of consensus mechanisms are *Proof-of-Work* (PoW), *Proof-of-Stake* (PoS), *Proof-of-Authority* (PoA), and *Practical Byzantine Fault Tolerance* (PBFT) [46].

There exist different types of BCs: *(i)* permissioned and *(ii)* permissionless. **Permissioned BCs**, such as BCs built on Hyperledger Fabric[2] and Substrate[3], are controlled by a single organization or consortium, which manages read and write permissions. **Private permissioned** chains keep data private to only authorized entities, while **public permissioned** BCs allow anyone to read data. **Permissionless BCs** can be accessed and used by anyone following the respective protocol.

Public BCs traditionally employ PoW consensus, where network participants solve difficult mathematical problems, which determines whether they can append blocks or not. Alternatively, they might implement PoS, where the ability to validate blocks is tied to the stake that an entity might have in the network (and as such, the incentive is to ensure that the chain works correctly).

Permissioned BCs prefer schemes such as PBFT or PoA. In PoA, a predefined set of authorities take turn to produce blocks. The other authorities then vote to include the proposed block, which is then appended to the chain. This has the advantage that it is less computationally intensive compared to the PoW mechanism.

3.2.1 Blockchains for REV

To understand why BCs are a good fit for REVs, let us briefly review how traditional voting works. Eligible voters cast their votes into ballot boxes. Once a ballot is in the ballot box, the ballot is considered final, and cannot be modified anymore or removed. These properties are called *immutability* and *append-only*. In traditional voting, there are multiple polling stations, geographically distributed, where voter can go to cast a vote. If someone wanted to rig a vote, it would require tampering with multiple ballot boxes, with a high risk of being detected. These relate to the *no single-point-of-failure*, *resilience*, *fault-tolerant*, and *decentralized* properties.

[2]https://www.hyperledger.org/
[3]https://www.substrate.io/

Essentially, what is needed is an immutable, append-only, distributed database. These properties can be found in a theoretical concept called a **public bulletin board** (PBB) [47]. A PBB is a component which provides an authenticated channel, providing transparency and verifiability to many schemes. It's main properties are [47]:

- Information can only be appended, not removed or modified.
- The board is public, *i.e.,* anyone may read the board.
- The board provides a consistent view to any viewer.

These properties, however, can be found in other database systems. What sets a BC apart is the addition of a consensus algorithm and byzantine failure tolerance, which provides a resilient, fault-tolerant system.

Among the different types of BCs, a PoA-based public permissioned BC fulfills all the requirements for a ballot box. A ballot box should provide a consistent view of its state, meaning that any participant of the network can verify that the chain is behaving correctly. It should be append-only and, once the data is included, be immutable. We do not wish for anyone to produce blocks, which in this case would mean including ballots for the final tally. This responsibility is given to a set of distinct authorities – the validators or sealers. Through the BC's consensus algorithm, trust is decentralized among the validators, ensuring that no single entity can interfere with the correctness and privacy of the vote. Additionally, decentralization also means that voter suppression is harder to achieve.

Public readable-BCs, however, also provide certain challenges, which drive key design decisions of any voting scheme. Transparency means that any action is visible to anyone in the network. In certain cases, for example, a voter could wish to hide the fact that they participated in a vote. Any data, which must remain confidential, must be protected by some means external to the BC. Many systems, for example, encrypt votes on a client device and make use of mix-nets or homomorphic encryption to protect the voter's privacy while counting votes.

3.2.2 Substrate

Parity's Substrate is a BC development kit for building custom BCs. It provides a modular architecture which enables developers to customize most aspects of a BC. Substrate's mental model abstracts a BC as a state machine with pluggable state transition functions. It provides, for example, ready-made modules for consensus, identity management, balances, governance, and smart contracts. Next to the ready-made modules, is the possibility to create custom modules called pallets. By developing a pallet it is possible to build custom logic directly into the BC runtime. Out of the box substrate chains use Aura, a round-robin consensus algorithm with an additional finality algorithm called Grandpa[4]. The prototype that accompanies this thesis, is built on Substrate, and more details can be found in Chapter 7.

[4]https://substrate.dev/docs/en/knowledgebase/advanced/consensus

Chapter 4

Related Work

This section presents the main approaches of REV systems and the major milestones over time. In literature, many systems were proposed. However, few were practically implemented, or have reached a mature production stage beyond research.

Chaum, in his seminal work, introduces secret-ballot voting protocols based on mix-nets [38]. The idea is to encrypt a ballot and send it through a mix, which would either re-encrypt or shuffle many ballots, until the original ballot becomes untraceable to an observer, at which point the ballot is decrypted and the plaintext is revealed for tallying.

In 1997, Cramer, Gennaro, and Schoenmakers [19] devised a multi-authority voting system based on homomorphic encryption to avoid needing to decrypt single votes, and threshold encryption to avoid giving any single entity the power to decrypt ballots.

A major milestone was the development of Helios [20]. Helios is one of the first practical systems published in 2008, which was used in real-world elections. Helios provides BS through public encryption. The vote is encrypted on the voter's client device, with the server's public-key, and sent to Helios. To provide verifiability, Helios implements Benaloh challenges [48] (see 4.1). Over the years, Helios was further developed to address the limitations and vulnerabilities detected by other researchers[1]. One of these advancements is Belenios [21], and its receipt-free extension BeleniosRF [49]. Cortier *et al.* [50] noticed that Helios does not protect against ballot stuffing in case of a malicious bulletin board. Thus, Belenios enhances Helios with the addition of a multi-authority scheme. BeleniosRF further builds on this, adding a trusted randomization authority.

Attempts to combine BCs and remote voting are much more recent. Liu and Wang [51] proposed in 2017 a scheme applicable to both permissioned and permissionless BCs. It relies on trust between a voter, an organizer, and an inspector. The voter encrypts the ballot using the organizer's public-key. Then the inspector signs it, using blind signatures. The voter can then cast it. The system assumes that no collusion occurs. The Open Vote Network [52] offers a self-tallying smart contract approach. Unfortunately, the scheme requires voters to be active after casting the ballots. Voters are also in charge of decrypting the votes, which decreases fairness, as the last voter can tally the final result before others.

[1] https://documentation.heliosvoting.org/attacks-and-defenses

The system was prototyped on the Ethereum BC and can only be used for small-scale elections [52].

The University of Zurich, developed Provotum [7] – a BC-based scheme – which is the starting point for this work. Provotum presents a trusted, centralized architecture backed by a BC, while Provotum 2.0 was developed to fully embrace decentralization, using a smart contract approach based on Ethereum.

4.1 Cast-as-Intended

There are two main approaches to Cast-as-Intended: *challenge-based* and *code-based*. Both are relevant mechanisms in the field of REV, and both have varying degrees of usability [53].

One *challenge-based* mechanism to provide ballot verifiability was proposed by Josh Benaloh, called *Cast-or-Challenge* or *Benaloh Challenge* [48]. A ballot can be challenged, in which case it is audited. An audited ballot must be discarded, and a new ballot must be generated. Otherwise, the audit could be used as a receipt for vote-selling, or compromise Ballot-Secrecy. Two different types of challenge were eventually proposed: either through decryption, or retracing the ballot's computation. In the first variant, the vote is decrypted to reveal the voter's choice. In the second, the cryptographic material used for the encryption is disclosed, and the vote is recomputed to show that the client device encrypted the voter's choice honestly. In either way, the voter's selection is exposed and the ballot must be discarded, as otherwise BS cannot be guaranteed. As the voting machine can not anticipate whether or not the voter will challenge the encryption, a compromised voting device can not know when to tamper with the ballot. *Cast or Challenge* mechanism are probabilistic, since it is not guaranteed that every voter will challenge the voting machine.

A different mechanism proposed later, was *Cast-and-Challenge* [54]. It is based on the same idea as Benaloh's, but does not require discarding the ballot, hence enforcing the challenge on each ballot created. The scheme is based on chameleon commitments to generate challengeable proofs without leaking privacy.

Code-based approaches have seen usage in Switzerland [55, 56] and Norway [57]. Each voter receives a code sheet via traditional mail, listing a check code for each voting option. Each voter receives a distinct code for each option. When a voter casts a vote, a code is either displayed on the voter's screen or sent via a second channel [57]. If the returned code matches the one on the sheet, the vote was cast correctly. Such systems usually rely on a secure server to store voter specific data.

4.2 Receipt-Freeness

Various schemes have been proposed in literature to achieve Receipt-Freeness. The concept was introduced by Benaloh and Tuinstra [42], who also proposed two protocols based on homomorphic encryption: a single-authority one and a multi-authority one. Both schemes assume a physical voting booth, providing a two-way secret channel between the voter and the authorities. Hirt [58] eventually showed that the first scheme did not guarantee Ballot-Secrecy, while the other was not receipt-free. Sako and Kilian [59], proposed a shuffle mix-net based approach, with a very high communication complexity in the mix, due to proof generation. Afterwards, Lee and Kim [60] and Baudron *et al.* [61], independently proposed the usage of a special entity to achieve Receipt-Freeness, called randomizer. The randomizer would act as a single node re-encryption mix-net, so that even a voter, using a biased-randomization source, would not be able to recreate the ballot as it appears on the public bulletin board. Lee and Kim's system required a trusted randomizer and was shown to not achieve Receipt-Freeness [33]. Baudron *et al.* [61], instead, proposed a system which did not rely on a trusted entity, but it is based on Paillier encryption and threshold cryptography, which requires increased computation for each authority on the voter's side. Finally, Hirt [33] presented an efficient ElGamal-based receipt-free scheme, which does not require additional trust in the randomizer. Unfortunately, later advances in the Fiat-Shamir heuristic (see Chapter 3) mean that a certain amount of trust in the randomizer still remains, as the proposed system applied a weak Fiat-Shamir transformation. Desmedt and Chaidos [34] showed an advanced version of the Helios ballot copying attack by Cortier and Smith [50], which allows a voter to copy another voter's vote. The authors proposed to blind the vote, as well as the validity proof, in order to make the ballot indistinguishable to the system. Interestingly, this attack, which can be applied only on low voter turnout, can be used to produce a receipt-free scheme.

In more recent works, BeleniosRF [49] designed a scheme, which achieves Receipt-Freeness using randomization and randomizable signatures to prove to the voter that the vote was not tampered with. Unfortunately, such a system relies heavily on the randomizer to act honestly, and presents an architecture which places all trust in the central voting authority.

One of the first receipt-free schemes based on a BC is due to Yu *et al.* [62], who proposes a scheme in which the chain re-encrypts the ballots it receives. However, the protocol requires a private permissioned BC, in order to protect the randomization used for the re-encryption. Such a system lowers the trust in any single component by distributing the responsibility of randomization on multiple nodes of a BC.

Chapter 5

Provotum 2.0 Security Analysis

The security of REV systems is fundamental for the fair execution of electoral and voting processes. In that sense, analyzing Provotum is a necessary and crucial step to validate the trust assumptions and assess risks, threats, and vulnerabilities.

The CSG group at the University of Zurich, developed Provotum 2.0 as a continuation of the Provotum project in 2020 [7]. Provotum 2.0 is a REV system based on a PoA BC, acting as its public bulletin board. It defines a multi-authority, homomorphic encryption voting scheme. The authors already identified some limitations that need to be mitigated before Provotum can be used for real-world votes and elections. Thus, this section conducts a security analysis of the scheme, in order to highlight any additional vulnerabilities and attack vectors, which will drive the improvements presented in Chapter 6.

5.1 Provotum 2.0 Design

In order to determine the risks to Provotum, we must define the scope of the system to protect, which also establishes the scope of the threat model. Figure 5.1 shows the design of the system and its stakeholders. One core component is the private PoA BC based on Ethereum, which serves as a PBB. The Provotum scheme is independent of the actual BC (in that case Ethereum), and could be replaced by any other BC which offers Smart Contract (SC) capabilities.

The main components of Provotum 2.0 can be summarized as follows:

- The **Identity Provider (IdP)** is a Trusted Third-Party (TTP) that verifies the identity and eligibility of a voter. After authenticating a voter, the IdP provides her with a one-time token which will prove that she is eligible to vote.
- The **Access Provider (AP)** is a first-party system acting on behalf of the voting authority, as a gatekeeper. In exchange for a valid token, it will fund the users' wallet with enough tokens to participate in the election. This decouples the authentication from the election process, increasing privacy: the IdP knows the identity of the voter

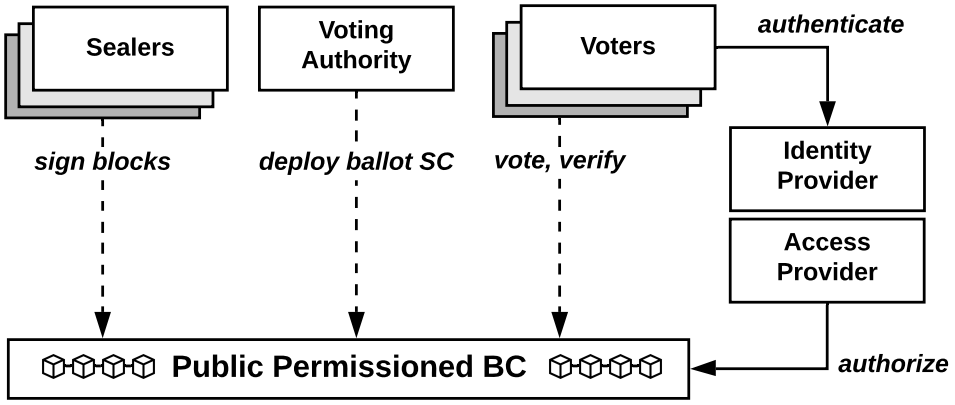


Figure 5.1: Provotum components

and the assigned token, while the AP knows the token and the wallet address. To link a ballot to a user, the IdP and the AP would have to collude and share their information.

- The set of **Sealers**, where a sealer is an entity running a node participating in the PoA BC, as a validator. Furthermore, Sealers participate in the distributed key generation and vote tallying.
- The **Voting Authority (VA)** is an entity coordinating the vote. The VA is responsible for the BC bootstrapping process (with the Sealers), deploying the voting SC, and opening and closing the vote.
- The **PoA BC** serves as an immutable PBB. PBBs are central component of REV systems. Equally important as the immutability aspect is the decentralization of storage and resilience.
- The set of **eligible voters** who participate by casting votes.
- The **general public** who is interested in having fair elections and votes, which can verify the data posted on the PBB at any time.

5.1.1 Identity Management in Provotum

Identity Management in Provotum is performed by three stakeholders: *(i)* the IdP, *(ii)* the AP, and *(iii)* the voter. The IdP is an independent TTP that verifies the voter's identity. The AP is a service run by the governing body of the vote (*e.g.,* canton, municipality, or federal government) which grants access to eligible voters. At the beginning of a vote, the IdP provisions and assigns random one-time tokens to each eligible voter. The IdP then sends the list of tokens to the AP. When a user authenticates herself to the IdP, she receives her token over an authenticated untappable channel.

The user can request access through the AP with her assigned token. Upon receiving tokens, the AP transfer ETH tokens to voter's address. This token is marked as used and the AP stores the wallet.

Analysis:
The described process assumes that the IdP is trusted. Otherwise, a malicious IdP could forge identities by minting more one-time tokens to give access to an adversary to rig the vote. The same holds for the AP who could fund wallets of users who have not been authenticated. Additionally, a collusion between IdP and AP makes linking an encrypted ballot to an identity trivial.

5.1.2 Vote setup and casting

The vote is composed of the following sequential steps:

1. Registration
2. Pairing
3. Key generation
4. Voting
5. Tallying

Registration

Each sealer creates its own Ethereum wallet, consisting of a public/private key pair. They start their applications and send their public-key to the voting authority. As mentioned in [63], it is important that these steps are conducted offline.

Analysis:
The paper mentions the importance of creating the keys offline. However, it does not specify that the public-key should be sent to the VA over an authenticated channel. In practice, this could be done physically or through a point-to-point connection.

Pairing

Each sealer fetches the BC's genesis configuration from the VA and start their BC nodes. The configuration provides the sealers and the AP with enough funds. Once all sealers are up and running, the VA deploys a smart contract. The SC is responsible of handling and storing vote's system parameters, questions, ballots and proofs, privileged addresses (VA, sealers, and AP who are not allowed to vote), and so on. Aside from storing ballots, it can also verify proofs and the final decryption of the tally.

From this point forward, the BC and the SC are operational, and further steps are conducted on the chain.

Analysis:
The protocol derives its security properties based on the properties of the underlying PoA BC. The nodes are considered semi-trusted, in that the network is resilient up to a certain threshold of byzantine actors. The threshold depends on the consensus algorithm in use.
Similar as before, this steps also assumes an authenticated channel to fetch the PoA configuration. Otherwise, MITM attacks would be possible to tamper with the configuration.

Distributed Key Generation

Each sealer generates an ElGamal key pair and a proof of knowledge of their own private-key. These are sent to the SC, which combines them into a single public-key: the combined public-key is used by voters to encrypt their ballots. By utilizing a cooperative decryption scheme, no single entity has power over the decryption: *i.e.,* if at least one sealer remains honest, the ballots remain confidential.

Analysis:
While the N/N cooperative scheme provides a very strong confidentiality guarantee against a byzantine party, it makes it also very brittle against denial of service. If a single sealer loses its key (*e.g.,* due to compromise), all ballots are effectively lost.

Voting

The voter constructs a ballot and encrypts it with the system's public-key. The voter also generates a proof of validity of the ballot. The encrypted ballot and proof are submitted directly to the SC to avoid trusting any intermediaries. If the proof is valid, the ballot is stored on the PBB.

Analysis:
The scheme claims to possess the Cast-as-Intended and Receipt-Freeness properties. However, these are not guaranteed. The claim of Receipt-Freeness holds only if the voter is honest. Although, Receipt-Freeness is the property of not being able to prove a vote, regardless of whether the voter is honest or not. A dishonest voter (*e.g.,* a bribed voter) can record the generated cryptographic material and, at any time, reconstruct the ballot such that it is exactly the same as the one stored on the PBB. In this scenario, the ballot becomes the receipt of the vote.
Additionally, Cast-as-Intended is only achieved under the assumption that the client's device remains honest. Although Cast-as-Intended was introduced as a way to protect against malicious voting devices, yet the assumption automatically negates the property.

Tallying

All sealer fetches the encrypted ballots and sums them homomorphically. The encrypted sum is then decrypted with the sealers' private-key. Every sealer also produces a proof of correct decryption. Each decrypted share is then sent back to the SC, which combines them to produce the final tally. A decrypted share does not leak any information under the DDH assumption.

Analysis:
The sealers share their decrypted shares publicly. A malicious sealer could wait for all other sealers to submit in order to compute the final tally on its own. Depending on the results, it could choose not to publish the last share, hence blocking the election.

Result

Once all sealers have submitted their decrypted shares, the VA can trigger the combination of shares. This combination determines the number of "*yes*" votes submitted. The entire process is run publicly on the SC and can be reproduced by anyone. Thus, no proof is necessary.

Analysis:
Executing this step on the BC does not bring any security improvements. The combination of the tally shares relies entirely on publicly available information and, as such, it can be computed and verified by anyone with access to the PBB. As a matter of fact, running the final decryption stage on the chain might affect its availability. Consider an election with millions of votes. The decryption would be long-running and computationally intensive, as it involves solving a discrete logarithm. This means that there is a high-risk of running out of gas during the decryption.

5.2 Threat Modelling

Given the context of online voting, the threat model considers highly motivated adversaries whose intention is either to tamper with the tally or to cause denial of service attacks in order to suppress voters. The adversary is computationally bound, *i.e.,* she cannot break assumptions based on the intractability of problems such as the DDH assumption [16].

5.2.1 Attack vectors and limitations

Possible threats to the scheme are presented as listed below, as a result of the analysis discussed previously.

ID	Title	Threat
T1	Lack of Cast-as-Intended	Cast-as-Intended is achieved only under the assumption that the voting device is not compromised and acts correctly, which automatically negates the property.
T2	Lack of Receipt-Freeness	A dishonest voter can easily reconstruct her own ballot to produce a receipt of her vote. It requires storing the randomization parameter used to encrypt the vote.
T3	Forced abstention	A coercer can stop a voter from voting by simply requiring the voter to submit her ether.
T4	Missing channel assumption	The protocol assumes the existence of an authenticated, untappable channel between sealers and VA. Without it, anyone could in practice register their wallet as a sealer or run a MITM attack when fetching BC's configs. This could be used to takeover the network using segmentation attacks/51% attack[1].
T5	Missing BC assumption	Reusage of the same BC for multiple votes could allow a voter to participate in the distributed key generation process. As shown in further findings (Table 5.6), due to implementation issues (not the protocol), the adversary could block the vote and force a retry or manipulate the tally.
T6	Attack of the clones	An attack to the consensus of PoA- or PoS-chains [64], which can be used to allow double spending. In this case, it could be used to prevent ballots from being included in the chain.
T7	Fault-tolerance	Due to the cooperative N/N decryption scheme, if a single sealer loses the key, all ballots are lost and the tally cannot be computed.
T8	Sealer-initiated Denial-of-Service	All sealers share their decrypted shares publicly and not at the same time. A malicious sealer could wait for all other sealers to submit, in order to compute the final tally on its own. Depending on the results, it could choose not to publish the last share, blocking the election.
T9	Decryption Denial-of-Service	The time required to decrypt the final tally is directly proportional to the number of "*yes*" votes received. This could potentially be long-running and cause a denial-of-service on the network. For an election with a high-turnout of voters, it would be better to compute the final tally offline, and then publish the results on the BC.
T10	Lack of ballot weeding	The protocol does not specify any protection against ballot reuse. As shown by Cortier and Smyth [50], if an adversary were to copy ballots of a user, this can be exploited to determine who voted for which candidate. However, this attack is only applicable on very low-scale votes.

Table 5.2: Provotum 2.0 scheme threats

[1] `https://en.bitcoin.it/wiki/Majority_attack`

As the IdP is considered a trusted entity, risks to identity management are listed separately, only for completeness.

ID	Title	Threat
T11	Session hijacking	The token issued by the IdP can be stolen or bought. This impedes an eligible citizen from voting, while allowing an adversary to vote, bypassing authentication.
T12	Token selling	The IdP token can be easily transferred to a vote buyer. Note that this is an issue with many remote voting systems, including postal vote.
T13	Voter impersonation	If a voter registers for electronic voting, but does not vote, the IdP can vote on the voter's behalf, by using her assigned token.
T14	Token re-use	The tokens provided from the IdP may not be reused in other elections. An adversary observing multiple elections could track usage of the same tokens to link votes to the same pseudonymous identity, which provides useful information for de-anonymisation through statistical analysis.
T15	IdP & AP collusion	Collusion of the IdP and AP would allow linking a ballot to an identity.
T16	IdP forges identities	A malicious IdP could forge identities by minting more one-time tokens.
T17	AP forges identities	A malicious AP could fund wallets of arbitrary users, allowing them to bypass authentication.
T18	AP voting	A malicious AP could create and fund its own wallets, allowing it to bypass authentication.

Table 5.4: Identity Management threats

The following two attacks are threats that were identified in the implementation, not the scheme.

ID	Title	Threat
T19	Missing authorization	The SC does not verify whether the public-key shares are coming from sealers or not. Anyone who sends a transaction with a valid public-key and proof, will take part in the cooperative scheme. The implicit assumption here is that no non-sealer user would have the funds necessary to submit transactions to the BC before the Voting phase. A colluding sealer or AP could transfer funds to a wallet address to allow someone else to send transactions. This could be used during the tallying phase to block the final tally decryption.
T20	Tampering	Due to an implementation flaw, it is possible for the last sealer to tamper with the final results. This could be detected by auditing all SC function calls, which is a plausible assumption to make, depending on the vote (low/high stakes). The flaw is that the SC does not compute the homomorphic sum but simply uses the one submitted by the sealers. In the case of honest sealers, the sum should always be the same. The SC also stores the last submitted homomorphic sum to use for the final decryption (as a way to not have to compute the homomorphic sum in the SC). A malicious sealer, given all decrypted shares, could attempt to find a different encrypted sum, such that when decrypted and combined, returns a final tally to the favor of some candidate.

Table 5.6: Implementation threats

Chapter 6

System Design of Provotum 3.0

This section describes Provotum 3.0. The updated system can be seen as an add-on to Provotum 2.0. This is reflected in the systems stakeholders as illustrated in Figure 6.1.

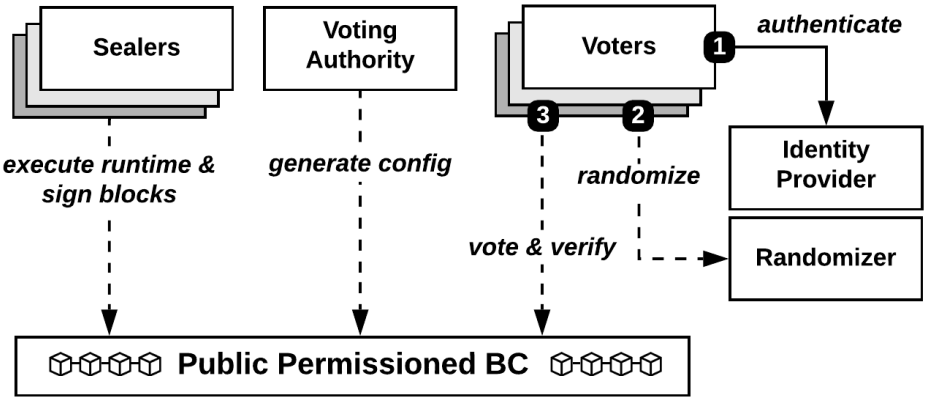


Figure 6.1: Provotum 3.0 architecture

In Provotum 2.0, the Access Provider entity was designed to provide voter's with the Ethereum (ETH) required in order to submit transactions to the chain. Owing to the replacement of the Ethereum BC with a Substrate-based one, the need for tokens is removed, thus the AP can also be removed from the system.

A small note on tokens most BCs introduce the concept of fees on each transactions as a mean of deterring abuse, such as spam and denial of service attacks – if an attacker has to spend money to attack a system, the incentive to do so is reduced. Furthermore, linking computational resources to a cost can be used to mitigate application layer denial of service attacks. In Provotum 2.0, ETH tokens were used as a proxy for authentication. In Provotum 3.0, even if tokens were used, it would not allow to cast votes, as all voters need to register themselves with the identity provider.

6.1 Stakeholders

Provotum defines a model consisting of N sealer authorities $(S_1, \ldots, S_N)$, M voters, one randomizer, one identity provider, one voting authority. Communications take place over a public bulletin board (PBB), represented by the Substrate BC. The PBB is an authenticated append-only channel. A threshold number of sealers t is needed in order to decrypt the final tally. For this thesis, we consider $N = t$, but replacing the distributed key generation algorithm with one that requires $t < N$ would be possible. The design assumes the usage of a BC, such as Substrate, in which the logic can be tied directly into the runtime. However, it can be replaced with a smart contract as in Provotum 2.0.

The main components of Provotum 3.0 can be summarized as follows:

- As with the previous iterations of Provotum, an **Identity Provider (IdP)** entity is introduced to the mix. It is a trusted third-party responsible for eligibility verification. Upon authentication, voters present the IdP with the public (blind) address of their wallet. The IdP signs the address and returns the signature to the voter. The voter then submits the signature to the BC, which verifies it and stores it. The signature will allow anyone to verify that a ballot, cast from an address, is approved by the IdP. Thanks to the use of blind signatures, the IdP will not be able to link a ballot to the real identity of a voter, hence providing a secure and privacy-preserving authorization mechanism.

- The **Randomizer** is a new entity added in Provotum 3.0. It's responsibility is to randomize (or blind) the voter's ballot, in order to achieve Receipt-Freeness. This is done by adding non-determinism to the process of creating a ballot. While the ElGamal encryption scheme includes a random nonce, a malicious voter can simply reuse the same nonce, allowing them to easily prove to a vote buyer how she voted. By adding a blinding factor, which is out of the voter's control, the voter is not able to reproduce the same ballot which is published on the PBB.

  The randomizer does not learn any of the votes it randomizes, since the blinding can be done without knowledge of the plaintext. In order to enforce that a voter randomizes the ballot, a randomizer's signature will be required to cast a ballot. To improve scalability and to avoid a single point of failure or that the randomizer prevents voters from participating, multiple randomizers can be used which could be operated by cantons, organizations, and NGOs. Additionally, a randomizer can be replaced with a government-issued hardware token.

  It is important to note, that collusion between the randomizer and a vote buyer or voter would trivially remove RF from the scheme.

  The randomizer does not need to communicate with any other part of the system.

- **Sealers** are entities responsible of running the BC validator nodes, as well as participating in the distributed key generation phase. Their public-key shares will be used to produce the election public-key, with which votes will be encrypted. Once the vote is over, the sealers will collaboratively decrypt the final tally. Due to the distributed nature of votes, each polling place, city, or canton could run a sealer.

- The **Voting Authority (VA)** is responsible of orchestrating the various phases of the vote. *(i)* It coordinates the BC bootstrap with the sealers, *(ii)* it stores the vote's topic and public-key generated through the distributed key generation, *(iii)* and it closes the vote, by signaling to the sealer to start tallying. The VA could be operated by a municipality, canton, or federal government.
- The **Substrate PoA BC** acts as the systems **PBB**. The BC implements the Provotum protocols directly into its runtime, exposing an API with which it is possible to submit transactions and read state tied to the voting process.
- **Voters** are eligible citizen who wish to participate in the democratic process. They communicate directly with the PBB to cast votes, thus removing the need to trust any single entity – *i.e.,* removing a single point of failure – and mitigating the risk of voter suppression.
- The **general public** is interested in having fair elections and votes, which can verify the data posted on the PBB at any time.

6.2 Voting Scheme of Provotum 3.0

Provotum follows in the steps of Cramer *et al.* [19]. A set of N talliers collaboratively generate a key pair, where the secret-key is shared among the authorities. Each voter encrypts her vote using the combined public-key and posts it to the public bulletin board. A vote encodes a "*yes*" or "*no*" as either a 1 or a 0, respectively. By homomorphically summing the encrypted votes, we obtain the encrypted amount of "*yes*" votes. The authorities jointly decrypt the sum, to reveal the plaintext value. By subtracting the amount of "*yes*" votes from the total amount of votes, we obtain the "*no*" votes. A special authority – the *Randomizer* – helps establish Receipt-Freeness by adding a source of mandatory randomization, which is never revealed to the voter.

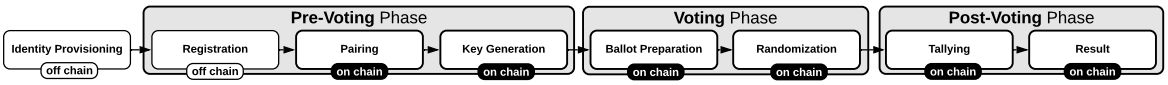


Figure 6.2: Provotum 3.0 scheme's phases

The voting scheme can be divided into three phases (Figure 6.2): *(i)* Pre-Voting/Setup Phase, *(ii)* Voting Phase, and *(iii)* Post-Voting Phase.

The scheme is largely similar to the one in Provotum 2.0. The key differences are:

- Voter's ballots are randomized before casting, making the process of generating a ballot non-deterministic.
- Identity Management: no Access Provider is needed, owing to the IdP that signs voter's addresses to allow them to vote.
- No smart contract, instead the logic is built directly into the chain.

6.2.1 Notation

The following sections make heavy use of various mathematical notation, which is briefly described here.

A ballot consists of a vote $v \in \{0, 1\}$. $E(v, \alpha)$ represents the ElGamal encryption of the vote v using a random nonce α. Σ is a DCP proof that the ballot contains either 0 or 1. Choosing a value r uniformly at random from a group of order q is denoted with $r \in_R \mathbb{Z}_q$. The homomorphic sum of two ciphertexts is denoted $e_1 \oplus e_2$. The multiplication of a scalar c and an encryption $e = (e_1, e_2)$ can be computed as $c \cdot e = c \cdot (e_1, e_2) = (e_1^c, e_2^c) \pmod p$, as demonstrated in Equation 2.19. H is a cryptographic hash function. The algorithm used is not important and can be easily swapped.

6.2.2 Identity Provisioning

The identity provider (IdP) is in charge of verifying the eligibility of a voter. It receives a list of identities from the relevant authorities. At any moment before the vote starts, it creates an RSA key pair to use for signing.

In order to register with the system, a voter creates a key pair for the BC. The public-key pk_v, is blinded as described in Chapter 3. The voter then sends the blind address pk_v, together with her credentials to the IdP. The IdP verifies the credentials and signs the address $\sigma' = \texttt{Sign}(pk_v)$. It then returns the signature σ' to the voter. The voter now unblinds the payload, revealing the signature for her public-key $\sigma = \texttt{Unblind}(\sigma')$ When the voter is ready to cast a ballot, she will send the signature σ together with the ballot to the BC. The BC will verify the origin pk_v of the ballot, the signature σ – if the signature matches, the BC will accepts the ballot as coming from an eligible voter.

For the RSA key pair $(sk_{idp}, pk_{idp}) = ((N, d), (N, e))$ and voter's public-key pk_v, we compute the blind signature as follows (all operations (mod N)):

Voter		**IdP**		**BC**
$r \in_R \mathbb{Z}_q$		knows d		
s.t. r, N coprime				
$m = H(pk_v)$				
$m' = m \cdot r^e$	$\xrightarrow{m'}$			
		$\sigma' = (m')^d$		
	$\xleftarrow{\sigma'}$			
$\sigma = \sigma' \cdot r^{-1}$			$\xrightarrow{\sigma}$	$m = H(pk_v)$
				$m \overset{!}{=} \sigma^e$

Figure 6.3: Voter-registration through blind signatures

The signature σ could in theory be sent at any moment between registration and casting the ballot. For simplicity, in the proof-of-concept this is done right after registration.

6.2.3 Pre-Voting Phase

During the **Pre-Voting Phase**, the sealers and VA coordinate to create the initial BC's configuration and public-keys to be used in the vote.

In the **Registration** step, each sealer node generates a Substrate wallet, consisting of a public/private key pair. The public-key/wallet address is sent to the VA which stores it. It is important that the keys are transmitted over an authenticated channel, otherwise an adversary could impersonate a sealer (see Chapter 5).

Based on the registered sealers, in the **Pairing** phase, the VA generates a genesis configuration file, which specifies the sealers as the validators of the PoA network, and the VA, which are allowed to create, open, and close votes. The VA publishes the genesis file and starts a BC node, to allow the other nodes to discover each other. The VA does not act as a validator node, although it could. Each sealer starts its own BC node to create the network and starts sealing blocks.

In contrast with Provotum 2.0, the voting logic is built into the BC runtime, and thus no deployment of a smart contract is needed. The Substrate chain exposes the necessary API to create a vote, set the public parameters required for encryption (refer to Chapter 3), cast ballots, verify proofs, and tally the votes.

The **Distributed Key Generation** step closes off the pre-voting phase. At the appropriate time, the VA creates a new vote, by sending a transaction including the vote's topic, and the public parameters in a transaction. Afterwards, each sealer fetches the public parameters to create an ElGamal key pair (pk_i, sk_i) and a Schnorr proof. They send the public-keys and the proof to the BC. The BC verifies the proof – if it is valid, it stores the public-key.

Once all validators have submitted their keys, the VA combines them into a single public-key pk_{voting} which is stored on the chain, and opens the vote.

6.2.4 Voting

The voting phase is divided in two steps. In the first one the voter chooses an option, encrypts it and generates a proof as in Provotum 2.0. In the second phase, the voter interacts with the randomizer to blind the encrypted ballot and to generate a blind DCP proof to provide Receipt-Freeness. Figure 6.4 illustrates the second phase.

Ballot Preparation

In the first step, the voter constructs a random encryption $e = E(v, \alpha)$, with randomness α, using the vote's public-key pk. The voter also generates a NIZKP, proving that v is in the correct range without leaking v. This first step is the same as in Provotum 2.0.

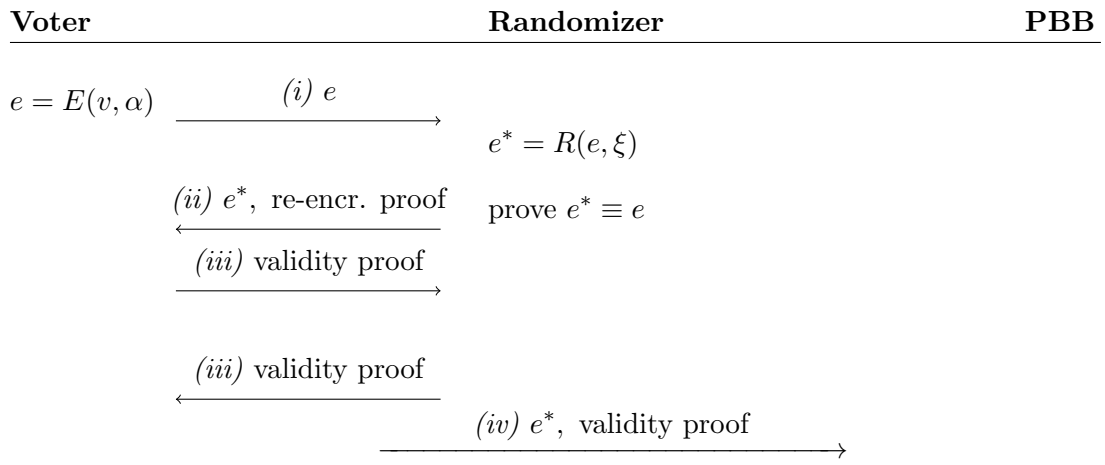


Figure 6.4: *(i)* the voter sends the encryption of her vote v to the randomizer. *(ii)* The randomizer blinds it and proves to the voter that the re-encryption still represents v. *(iii)* Both voter and randomizer interactively generate a proof for the re-encrypted vote. *(iv)* The re-encryption and the validity proof are sent to the PBB.

The second step is an interactive protocol between voter and randomizer. The voter sends the vote $e = E(v, \alpha)$ and the NIZKP to the randomizer. The randomizer first verifies that the ballot is valid, then it proceeds with the randomization protocol.

Vote Randomization

The randomizer can randomize (or blind) the ballot using the property that, by homomorphically adding an encryption of 0, the contents of the encrypted ballot do not change [33, 34]. Formally, for a ballot $e = E(v, \alpha) = (c, d) = (g^\alpha,\ h^\alpha g^v)$, a randomization e^* with nonce ξ is defined by the randomization function $R(e, \xi)$, which can be computed as

$$\begin{aligned} e^* = R(e, \xi) &= E(0,\ \xi) \oplus E(v,\ \alpha) \\ &= (g^\xi,\ h^\xi) \oplus (g^\alpha,\ h^\alpha g^v) \\ &= (g^\xi g^\alpha,\ h^\xi h^\alpha g^v) \\ &= (g^{\xi+\alpha},\ h^{\xi+\alpha} g^v) \\ &= E(v,\ \xi + \alpha) \end{aligned} \tag{6.1}$$

The randomization performs a re-encryption of the same vote v. It is indistinguishable if ξ is chosen at random, under the DDH assumption. Additionally, it is not reproducible, if the randomizer behaves honestly and uses a different random parameter ξ for each ballot randomization.

In order to prove to the voter that the returned ballot e^* does not encrypt a different vote as the one intended, the randomizer must produce a ZKP. The simplest way to do this is to prove the equivalence of two discrete logarithms as $E(0, \alpha) = e^* \ominus e$. The meaning of $\ominus$ is "*Homomorphically add the inverse of the value*", *i.e.*, $e' \ominus e = e' \oplus e^{-1}$.

Definition 6.2.1 (Proof of Re-encryption) *A proof of re-encryption of an encrypted value e is a ZKP of equivalence of $E(0,\alpha) = e^* \ominus e,\ \alpha \in \mathbb{Z}_q$, where e^* is a re-encryption of e, i.e., $e^* = R(e,\xi)$ [33].*

Given a voter's ballot $e = E(v,\alpha)$, a randomization $e^* = R(e,\xi) = E(0,\xi) \oplus e$, a public-key pk, and public parameters (p,g,q), we want to prove that $e^- = e^* \ominus e$ equals some encryption of 0, *i.e.*, $e^- = E(0,\xi)$, without leaking ξ [33].

Randomizer		**Voter**
knows $e^-,\ \xi$		knows e^-
s.t. $e^- = E(0,\xi)$		
$\alpha' \in_R \mathbb{Z}_q$		
$e' = E(0,\alpha')$		
	$e' \longrightarrow$	
		$c \in_R \mathbb{Z}_q$
	$\longleftarrow c$	
$\beta = c \cdot \xi + \alpha'$		
	$\beta \longrightarrow$	
		$E(0,\beta) \stackrel{!}{=} c \cdot e^- \oplus e'$

Figure 6.5: Proof of $E(v,\alpha) = e \equiv e^* = R(e,\xi)$

Despite this proof, one last step is necessary in order to ensure Receipt-Freeness. In fact, if the proof were sent like this to the voter – the randomization proof, together with the original and re-randomized ballots – would be a receipt of the vote. For this reason, we use designated-verifier proofs, implemented using Schnorr's identification scheme [30].

Definition 6.2.2 (Designated-Verifier Proof of Re-encryption) *A designated-verifier proof of re-encryption is a ZKP of re-encryption, composed of a logical disjunction of a Schnorr Proof, together with a re-encryption proof. The proof can only be verified by the voter, identified by the public-key pk_{voter} [33].*

Given all the parameters from the previous definition, together with the public-key $pk_{voter} = g^{sk_{voter}}$, we wish to prove that $e^- = e^* \ominus e$ is equivalent to some encryption of 0, *i.e.*, $e^- = E(0,\xi)$, without leaking ξ [33].

Randomizer		**Voter**
knows $e^-,\ \xi$		knows e^-
s.t. $e^- = E(0, \xi)$		
$\alpha' \in_R \mathbb{Z}_q,\ e' = E(0, \alpha')$		
$c_2 \in_R \mathbb{Z}_q,\ s_2 \in_R \mathbb{Z}_q,$		
s.t. $t_2 = g^{s_2} \cdot pk_{voter}^{-c_2}$	$\xrightarrow{e',\ t_2}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_q$
$c_1 = c - c_2 \pmod q$		
$\beta = c_1 \cdot \xi + \alpha'$	$\xrightarrow{c_1,\ c_2,\ \beta,\ s_2}$	$c \stackrel{!}{=} c_1 + c_2 \pmod q$
		$E(0, \beta) \stackrel{!}{=} c_1 \cdot e^- \oplus e'$
		$g^{s_2} \stackrel{!}{=} pk_{voter}^{c_2} \cdot t_2$

Figure 6.6: Interactive designated-verifier proof of re-encryption

The algorithm can be turned into a non-interactive proof via the Fiat-Shamir heuristic, with $c = H(e', e^-, t_2)$.

Randomizer		**Voter**
knows $e^-,\ \xi$		knows e^-
s.t. $e^- = E(0, \xi)$		
$\alpha' \in_R \mathbb{Z}_q,\ e' = E(0, \alpha')$		
$c_2 \in_R \mathbb{Z}_q,\ s_2 \in_R \mathbb{Z}_q,$		
s.t. $t_2 = g^{s_2} \cdot pk_{voter}^{-c_2}$		
$c = H(e', e^-, t_2)$		
$c_1 = c - c_2 \pmod q$		
$\beta = c_1 \cdot \xi + \alpha'$	$\xrightarrow{e',\ t_2,\ c_1,\ c_2,\ \beta,\ s_2}$	$c = H(e', e^-, t_2)$
		$c \stackrel{!}{=} c_1 + c_2 \pmod q$
		$E(0, \beta) \stackrel{!}{=} c_1 \cdot e^- \oplus e'$
		$g^{s_2} \stackrel{!}{=} pk_{voter}^{c_2} \cdot t_2$

Figure 6.7: Non-interactive designated-verifier proof of re-encryption

With this we have convinced the voter, that her vote has not been altered. However, we still need to prove to the rest of the system that the randomized ballot is correct.

Ballot Proof Randomization

A validity proof for a randomized ballot $e^* = R(E(v,\alpha),\xi)$ is a non-interactive proof, where e^* contains a valid vote $v \in \{0,1\}$. Both voter and randomizer cannot produce the proof alone, as neither know all parameters that formed e^*. For this reason, voter and randomizer must collaborate to create a NIZKP, which the rest of the system can verify. The algorithm can be divided in two steps: *(i)* voter and randomizer collaborate to create a divertible proof for e [32, 33], then *(ii)* the randomizer diverts this proof into a valid proof for e^* [33, 34]. Given a vote $v \in \{0,1\}$, a ballot $e = (c,d) = E(v,\alpha)$, a re-encryption $e^* = R(e,\xi)$, a public-key h, and public parameters (p,g,q), we interactively produce a divertible proof of validity as in Figure 6.8.

Voter		**Randomizer**
knows $e,\ \alpha,\ v \in \{0,1\}$		knows e
s.t. $e = E(v,\alpha) = (c,d)$		
$\lambda\ = 1 - v$		
$w,\ c_\lambda,\ s_\lambda \in_R \mathbb{Z}_q$		
$a_v = g^w \pmod p$		
$b_v = h^w \pmod p$		
$a_\lambda = g^{s_\lambda}/c^{c_\lambda} \pmod p$		
$b_\lambda = h^{s_\lambda}/(d/g^\lambda)^{c_\lambda} \pmod p$		
For $i = 0,1$, set values a_i, b_i	$\xrightarrow{a_0, b_0, a_1, b_1}$	$\Delta_0,\ \Delta_1,\ k_0,\ k_1 \in_R \mathbb{Z}_q$
		For $i = 0,1$
		$A_i = a_i g^{k_i}/c^{\Delta_i} \pmod p$
		$B_i = b_i h^{k_i}/(d/g^i)^{\Delta_i} \pmod p$
		$t\ = H(A_0, B_0, A_1, B_1)$
$c_v = C - c_\lambda \pmod q$	$\xleftarrow{C}$	$C\ = t - \Delta_0 - \Delta_1 \pmod q$
$s_v = w + \alpha \cdot c_v \pmod p$	$\xrightarrow{c_0, s_0, c_1, s_1}$	verify, for $i = 0,1$
		$C \stackrel{?}{=} c_0 + c_1 \pmod q$
		$a_i \stackrel{?}{=} g^{s_i}/c^{c_i} \pmod p$
		$b_i \stackrel{?}{=} h^{s_i}/(d/g^i)^{c_i} \pmod p$
		If valid, set
		$C_i = c_i + \Delta_i \pmod p$
		$S_i = s_i + k_i \pmod p$
	$\xleftarrow{\Sigma}$	$\Sigma\ = (A_0, B_0, A_1, B_1, C_0, S_0, C_1, S_1)$

Figure 6.8: Ballot proof (DCP) randomization

If $v = 0$, set: $a_0 = a_v,\ b_0 = b_v,\ a_1 = a_\lambda,\ b_1 = b_\lambda$
If $v = 1$, set: $a_1 = a_v,\ b_1 = b_v,\ a_0 = a_\lambda,\ b_0 = b_\lambda$; do the same with c_i and s_i

At this point, the tuple $\Sigma = (A_0, A_1, B_0, B_1, C_0, C_1, S_0, S_1)$ is a valid divertible non-interactive proof for e – *i.e.,* Σ verifies that e is valid, but can also be modified to verify e^* instead. It is also trivial to see that Σ is in essence a normal DCP proof, created collaboratively by two parties without disclosing their respective secrets.

A diverted proof for the re-encryption e^* can be computed by modifying the response component of Σ. For a given ballot e, a proof $\Sigma = (A_0, A_1, B_0, B_1, C_0, C_1, S_0, S_1)$, a re-encryption nonce ξ, and a re-encryption e^*, the tuple Σ^* defines a valid proof for e^*, with

$$\Sigma^* = (A_0, A_1, B_0, B_1, C_0, C_1, S_0 + C_0\xi, S_1 + C_1\xi) \tag{6.2}$$

The randomizer will thus return Σ^* instead of Σ to the voter. Proofs of correctness and soundness are given in Hirt [33] and Desmedt and Chaidos [34]. Of special importance for Receipt-Freeness is the indistinguishability property. The blinded proof is indistinguishable from a valid proof generated independently. If the parameters $\Delta_0, \Delta_1, k_0, k_1$ are chosen uniformly at random, the probability of a dishonest voter to link the diverted to the original proof, is negligible.

The keen eye will notice that only the commitments A_i, B_i are hashed in the protocol in Figure 6.8. This means that the protocol uses a weak Fiat-Shamir transformation. Unfortunately, due to the fact that the proof is generated for the encryption e, and then diverted to e^*, the statement (the encryption) cannot be part of the hash. Should it be the case, then after being diverted, the proof would not be correct (as the ciphertext has changed) and the computation of the hash would result in a different challenge being produced.

Since the randomized ballots are indistinguishable from any ballot, which the voter could have generated on her own, a voter could choose to not randomize the ballot. To enforce the protocol, the randomizer must provide a digital signature of the ballot (e^*, Σ^*), which the BC verifies before accepting a ballot.

Vote Casting

The final ballot to be cast is then represented by the tuple

$$(e^*, \Sigma^*) = (R(e, \xi), (A_0, A_1, B_0, B_1, C_0, C_1, S_0 + C_0\xi, S_1 + C_1\xi)) \tag{6.3}$$

The voter casts the tuple directly to the PBB, avoiding the need to trust any intermediaries. The PBB verifies that the address (from which the ballot was cast) matches the signature of a known voter, registered in the voter-registration phase. If it does, and the address was not used to cast any votes yet, it verifies the Σ^* via the normal DCP proof, as described in Chapter 3. If the proof is also valid, the ballot is stored for tallying.

The BC rejects any ballot which is an exact copy of either the ballot or proof, to avoid copying attacks [50]. An adversary could in theory blind the ballot such that it is a copy, but indistinguishable. However, it cannot generate an indistinguishable proof, due to the lack of the nonce used to encrypt the original ballot.

6.2.5 Post-Voting Phase

The VA can close the vote and proceed to the tallying phase. In this phase, each sealer fetches the encrypted ballots from the BC, it sums them homomorphically (as previously described), and produces a decrypted share of the final tally. All N shares are necessary to find the plaintext, so the share does not leak any information. In order to prove to have decrypted correctly, each sealer produces a proof that the share was decrypted with their respective private-key sk_i belonging to the public-key pk_i, which was submitted to the chain in the DKG phase.

Every sealer then submits the decrypted share together with the proof, which the BC can verify and store, if valid.

Once all sealers have submitted their share, the final tally can be revealed. It is important to stress that the threshold encryption scheme requires all N sealers to act honestly. If any sealer acts maliciously, the tally cannot be computed.

The final plaintext value will reveal the amount of "*yes*" votes, cast for a specific topic. By simple subtraction, the number of "*no*" votes can be computed:

$$|votes_{yes}| = |votes_{total}| - |votes_{yes}| \tag{6.4}$$

6.3 Summary

Provotum 3.0 employs many cryptographic primitives. Table 6.1 shows a concise mapping of the algorithms to Provotum's phases and steps, described in detail in Section 6.2.

Phases	**Steps**	**Primitives**
IdM	Voter registration	RSA blind signatures
Pre-voting	Registration	Substrate key-generation
	Pairing	PoA
	DKG	N/N Threshold ElGamal key-generation, Schnorr
Voting	Ballot preparation	ElGamal, DCP
	Randomization	ElGamal re-encr., DCP, Designated-verifier re-encr.
Post-voting	Tallying	ElGamal, Homomorphic sum, Chaum-Pedersen

Table 6.1: Cryptographic primitives applied in Provotum 3.0

Chapter 7

Implementation

Along this thesis, a prototype was developed implementing the design proposed in Chapter 6. All relevant source code has been published on Github under the Provotum organization[1]. The structure of packages is illustrated in Figure 7.1.

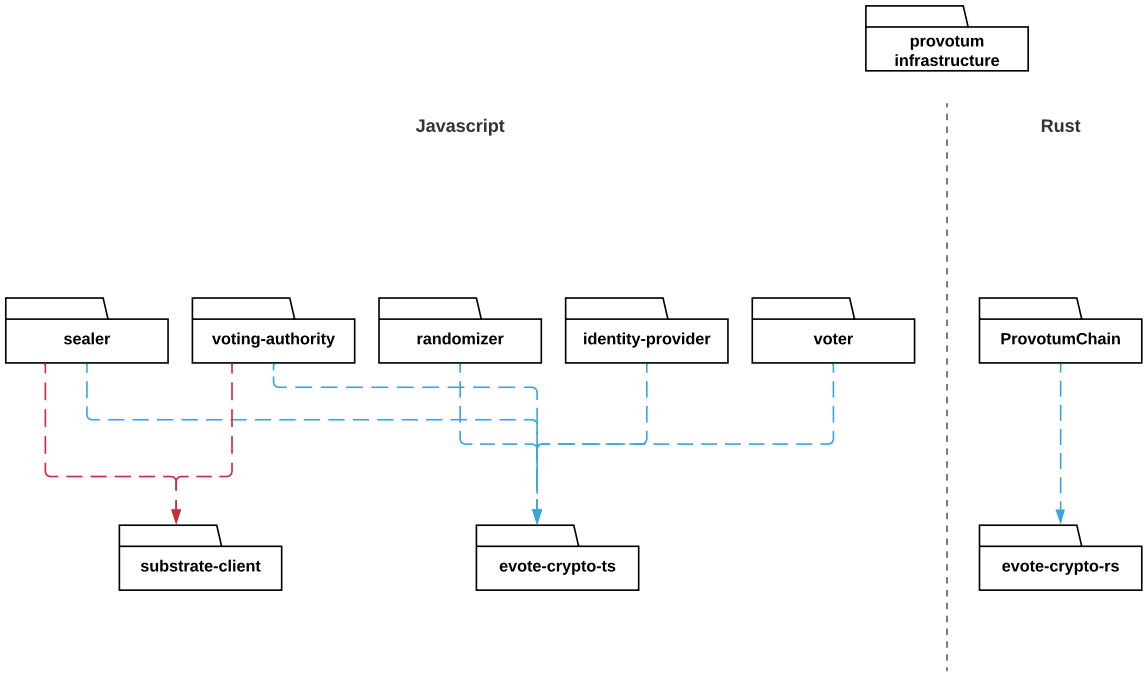


Figure 7.1: Provotum 3.0 package structure

The `ProvotumChain` package stores the blockchain (BC) source code, which was developed using Substrate[2] – a modular framework that can be used to create purpose-built BCs by defining custom logic right into the BC itself.

Substrate is developed in Rust[3] and focuses on developer experience, offering various

[1] https://github.com/provotum
[2] https://substrate.dev/
[3] https://www.rust-lang.org/

mature Typescript[4] client-libraries for decentralized application development[5]. Libraries in other languages are currently in development. However, as of the writing of this thesis, they are not ready yet. As such, Provotum 3.0 backend applications are implemented in Typescript.

The `voter` package is a frontend application, developed in Javascript and React[6]. The `voter` communicates with the `Randomizer` and the `Identity Provider` via HTTP, while it communicates with the BC using the Polkadot.js library[7] over WebSockets [65]. `Sealer`, `Voting Authority`, `Randomizer`, and `Identity Provider` are backend-only, nodejs applications, written in Typescript. All Javascript and Typescript applications depend on the `evote-crypto-ts` package, which implements all crypto primitives described in Chapter 2. The library `evote-crypto-rs` is a Rust port of the Typescript package, developed for usage from `ProvotumChain`.

Finally, the `Provotum Infrastructure` repository holds Terraform[8] and Ansible[9] scripts. It can be used to deploy Provotum on a single server or on a fleet of servers.

7.1 ProvotumChain

ProvotumChain is the package storing the source code for all BC code, written using Substrate. This includes the actual BC runtime, as well as the custom, voting logic. Substrate models a BC as a state machine, with a customizable state transition-function. Custom logic, can be written into modules called *pallets*, with which the BC runtime can be composed. Each pallet can define functionality and external interfaces, which can be called from client applications. Pallets are written in Rust, a general purpose language, which makes it easier to integrate with other libraries in the ecosystem. However, Rust and Substrate have a steep learning curve, and in order for the chain to compile, pallet code must be compilable into WebAssembly[10] (Wasm). This limits and drives many of the design choices when building an application on Substrate.

Substrate makes heavy use Rust macros[11] to generate the "glue code" necessary to tie the pallet into the runtime, at compile time. Many of the following concepts have a Solidity Smart Contract equivalent. The `decl_storage` macro is used to define the pallet's persistent storage, similarly to a smart contracts field in Solidity. The macros `decl_event` and `decl_error` provide a way to define the events and errors emitted by the pallet. Finally, `decl_module` defines which callable functions are exposed and it orchestrates actions that this pallet takes during block-execution (Listing 1)[12].

[4] https://www.typescriptlang.org/
[5] https://github.com/polkadot-js/
[6] https://reactjs.org/
[7] https://github.com/polkadot-js/
[8] https://www.terraform.io/
[9] https://www.ansible.com/
[10] https://webassembly.org/
[11] https://doc.rust-lang.org/1.7.0/book/macros.html
[12] https://substrate.dev

```
// Imports and Dependencies
use support::{decl_module, decl_event, decl_storage, decl_error}

// Runtime Configuration Trait: runtime types and constants
pub trait Trait: system::Trait { ... }

// Storage
decl_storage! { /* --snip-- */ }

// Events
decl_event! { /* --snip-- */ }

// Errors
decl_error! { /* --snip-- */ }

// Callable Functions
decl_module! { /* --snip-- */ }
```

Listing 1: Structure of a Substrate pallet

WebAssembly compatibility For future Substrate developers, a good rule-of-thumb on whether a Rust package (or crate as they are called) is not Wasm compatible and will thus not work with Substrate, is to check whether the code makes use of any I/O and dynamic allocators (*e.g.,* variable sized collections and strings). In general, crates marked as `no_std` will be compatible and they will advertise this on their repository or on the crates directory[13].

7.1.1 Data Model

Substrate stores data in a key-value data store. It combines a traditional database with a Merkle-tree-structure for integrity. It then provides high-level abstractions to facilitate working with persistent data. The use of Wasm directly influences how data is written and read. It defines its own codec, called SCALE: any value that is to be persisted must be compatible with SCALE[14]. In general, Rust primitives and byte-vectors of arbitrary size are compatible and can be written to storage. Substrate allows defining custom data structures, but the fields they define must follow the mentioned restriction. This means that more complex data-types have to be marshalled for saving, and demarshalled to operate on it. As an example, Provotum makes use of the `BigInteger` crate for operations on arbitrarily large integers, which are necessary in order to achieve a secure key-size (*e.g.,* 2048-bit and upwards). In contrast, Provotum 2.0, which is based on Ethereum and Solidity, is limited to 256-bit keys [7]. `BigInteger` instances cannot be directly stored on the chain. For this reason, public-keys, ciphertexts, and proofs are sent over the wire as hex strings, converted into bytes for storage, and wrapped in `BigIntergers` for cryptographic computations.

The following section will discuss some of the most important storage items.

[13] https://crates.io/
[14] https://substrate.dev/docs/en/knowledgebase/runtime/storage

Voter-Registration

The `identityProviderPublicKey` field (Listing 2) stores the IdP's public-key used to verify voter-eligibility and can only be set by a VA. Once a voter obtains and submits a signature for its address, if it is valid, the voter's address will be added to the `Voters` map. When a voter then casts a ballot, the chain will first verify if the address the ballot was cast from is included in `Voters`.

```
decl_storage! {
  trait Store for Module<T: Trait> as ProvotumModule {
    /// Stores the Identity provider's public-key to verify signatures.
    pub IdentityProviderPublicKey: Option<RsaPublicKey>;
    /// Maps an address to an IdP signature.
    Voters get(fn voters): map hasher(blake2_128_concat) T::AccountId => Vec<u8>;
  }
}
```

Listing 2: Voter-registration storage items

Vote Setup

The `VotingAuthorities` and `Sealers` storage items self-evidently hold the addresses of the voting authorities and sealers. They are defined in the genesis configuration file and, through the `config()` extension, they are initialized and made available to the pallet automatically. They are implemented in the Provotum pallet, in order to enforce authorization checks. The VA can create an election, which can comprise of multiple subjects (or topics). These are stored in the `Elections` and `Subjects` items. An election persists the public-key of the VA who created it, a title, its current vote-phase, and the public cryptographic parameters, as illustrated in Figure 7.2. A subject simply saves a question, encoded in bytes. Finally, `PublicKeyShares` and `PublicKey` store the sealers' shares and the computed vote public-key, as shown in Listing 3.

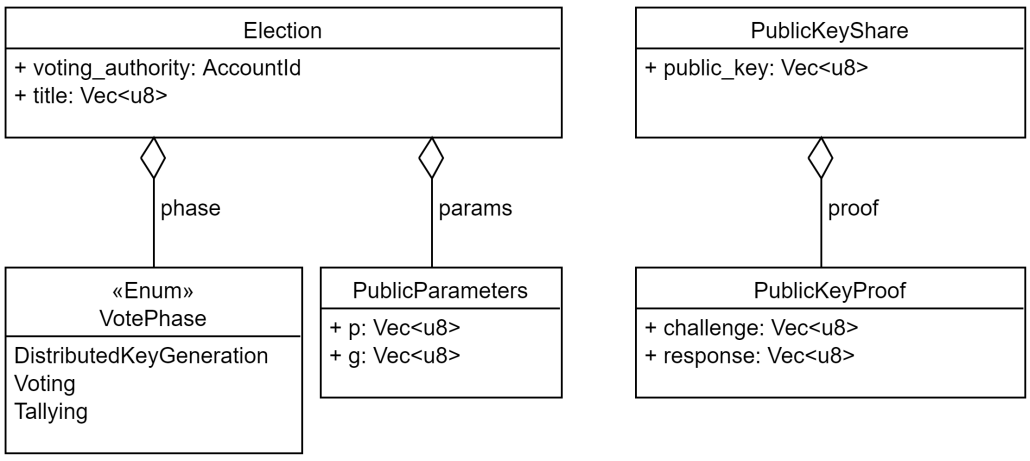


Figure 7.2: Election structure

```
decl_storage! {
  trait Store for Module<T: Trait> as ProvotumModule {
    /// Voting authorities
    VotingAuthorities config(): Vec<T::AccountId>;
    /// Sealer authorities
    Sealers config(): Vec<T::AccountId>;
    /// Stores the public-key of a sealer together with its proof.
    PublicKeyShares: map hasher(blake2_128_concat) ElectionId => Vec<PublicKeyShare>;
    /// Maps a vote to a public-key, used to encrypt ballots.
    PublicKey: map hasher(blake2_128_concat) ElectionId => Vec<u8>;
    /// Maps ElectionId to an election.
    Elections: map hasher(blake2_128_concat) ElectionId => Election<T::AccountId>;
    /// Maps ElectionId to subjects.
    Subjects: map hasher(blake2_128_concat) ElectionId => Vec<(SubjectId, Vec<u8>)>;
  }
}
```

Listing 3: Vote setup storage items

Voting

When the election is in the voting-phase, voter can cast their ballots as a vector of tuples of the subject the vote is for: a ciphertext and a proof (Figure 7.3). The ballot is verified by checking, in turn, each encrypted vote and its proof. If all are valid, they are stored. Otherwise, the entire ballot is rejected, as indicated in Listing 4#L15. This vector is then split in one vote for each subject and saved in the `Votes` storage item (Listing 4). This simplifies the tally at the end by already separating ballots by subject.

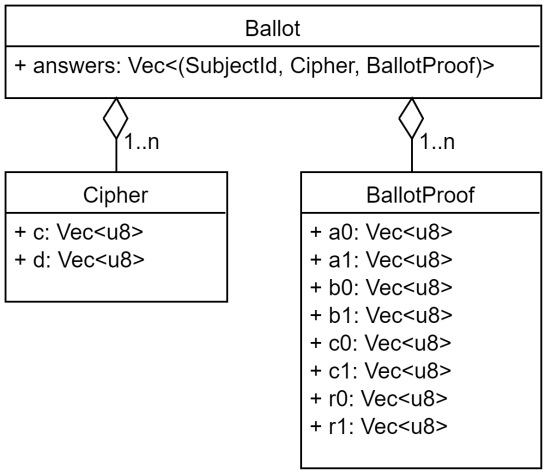


Figure 7.3: Ballot structure

```
decl_storage! {
  trait Store for Module<T: Trait> as ProvotumModule {
    /// Maps an account and a vote to a ballot.
    Ballots: map hasher(blake2_128_concat) (ElectionId, T::AccountId) => Ballot;
    /// Maps SubjectId to a list of encrypted votes.
    Votes get(fn votes): map hasher(blake2_128_concat) SubjectId => Vec<Cipher>;
  }
}

fn verify_and_store_ballot(origin: &T::AccountId, election_id: &ElectionId, ballot:
↪  Ballot) -> Result<(), Error<T>> {
  let unique_id: Vec<u8> = origin.encode();
  let public_key: Vec<u8> = PublicKey::get(&election_id);
  let params: PublicParameters = Self::get_params(election_id);
    match ballot.verify(&params, &public_key, &unique_id) {
      Ok(_) => Ok(Self::store_ballot(origin, election_id, ballot)),
      Err(_) => Err(Error::<T>::PublicKeyProofError)
    }
}

fn store_ballot(origin: &T::AccountId, vote_id: &ElectionId, ballot: Ballot) {
  Ballots::<T>::insert((&vote_id, origin), ballot.clone());
  for (subject_id, cipher, _) in ballot.answers.iter() {
    let mut votes: Vec<Cipher> = Votes::get(&subject_id);
    votes.push(cipher.clone());
    Votes::insert(&subject_id, votes);
  }
}
```

Listing 4: Storing ballots and separating vote ciphertexts by subject to simplify tallying.

Tally

At the appropriate time, the VA can start the counting-process by calling the extrinsic. This will homomorphically add all ballots and store them in the `EncryptedTallies` map (Listing 5). In turn, each sealer will fetch the computed sum, decrypt it, and submit it with the proof of correct decryption: if the proof is valid, the decrypted share is saved. Once all shares have been submitted, the VA can trigger the final decryption-step to reveal the plaintext count of "*yes*" votes. The results are persisted in the `Tallies` map by election. The homomorphic-sum and the final decryption-step do not require any proof, as they can be verified with all the publicly available information.

7.2 Technical Limitations

The prototype shows minor technical limitations, which can be addressed in future work. For example, certain checks are not enforced: the randomizer signature is not actually requested, hence a voter may possibly cast multiple ballots.

```
decl_storage! {
  trait Store for Module<T: Trait> as ProvotumModule {
    /// Homomorphic-sum of encrypted votes
    EncryptedTallies: map hasher(blake2_128_concat) SubjectId => Cipher;
    /// Decrypted shares
    DecryptedShares: map hasher(blake2_128_concat) SubjectId => Vec<Vec<u8>>;
    /// Maps a vote to a list of results: [topic, yes, no, total].
    Tallies: map hasher(blake2_128_concat) ElectionId => Vec<Tally>;
  }
}

/// Sets tallying-phase, computes homomorphic-sum.
fn set_tallying_phase(origin, vote_id: ElectionId) {
  let who = ensure_signed(origin)?;
  ensure_voting_authority::<T>(&who)?;
  let phase = VotePhase::Tallying;
  set_phase::<T>(&vote_id, phase.clone());
  Self::sum_ballots(&vote_id);
  Self::deposit_event(RawEvent::VotePhaseChanged(vote_id, phase));
}

/// Stores a decrypted share.
fn submit_decrypted_share(origin, vote_id: ElectionId, subject_id: SubjectId, share:
↪  Vec<u8>, proof: DecryptedShareProof) {
  let who = ensure_signed(origin)?;
  ensure_sealer::<T>(&who)?;
  Self::verify_and_store_decrypted_share(who.clone(), vote_id, subject_id.clone(),
↪  share.clone(), proof)?;
  Self::deposit_event(RawEvent::DecryptedShareSubmitted(who, subject_id, share));
}

/// Combines decrypted shares into a final plaintext-tally.
fn compute_tallies(origin, vote_id: ElectionId) {
  let who = ensure_signed(origin)?;
  ensure_voting_authority::<T>(&who)?;
  Self::combine_decrypted_shares(&vote_id);
}
```

Listing 5: Tallying

Chapter 8

Evaluation

This chapter evaluates the proposed design and prototype. First, we analyze the e-voting security properties of the novel design and prove that Receipt-Freeness is achieved, while maintaining the remaining security properties. Afterwards, in Section 8.2, we evaluate the implemented prototype in terms of scalability, showing that it is possible to scale to nation-wide votes.

8.1 Security Analysis

The following table is a revision of the threats, firstly reported in Section 5.2, which are mitigated through the new design. The main focus of this thesis was to bring Receipt-Freeness to the Provotum scheme.

ID	Title	Threat	Mitigation
T2	Lack of Receipt-Freeness	A dishonest voter can easily reconstruct her own ballot to produce a receipt of her vote. It requires storing the randomization parameter used to encrypt the vote.	The introduction of the randomizer adds Receipt-Freeness to the scheme.
T3	Forced abstention	A coercer can stop a voter from voting by simply requiring the voter to submit her ether.	Tokens are not present in the Substrate chain.
T4	Missing channel assumption	The protocol assumes the existence of an authenticated, untappable channel between sealers and VA. Without it, anyone could in practice register their wallet as a sealer or run a MITM attack when fetching the BC's configuration. This could be used to takeover the network using segmentation attacks/51% attack.	Addressed in design.

T5	Missing BC assumption	Reusage of the same BC for multiple votes could allow a voter to participate in the distributed key generation process. As shown in further findings (Table 5.6), due to implementation issues (not the protocol), the adversary could block the vote and force a retry or manipulate the tally.	Addressed in design.
T10	Lack of ballot weeding	The protocol does not specify any protection against ballot reuse. As shown by Cortier and Smyth [50], if an adversary were to copy ballots of a user, this can be exploited to determine who voted for which candidate. However, this attack is only applicable on very low-scale votes.	Added ballot weeding of any ballot, which is either an exact copy of the ciphertext or a DCP ZKP.
T11	Session hijacking	The token issued by the IdP can be stolen or bought. This impedes an eligible citizen from voting, while allowing an adversary to vote, bypassing authentication.	Removed use of the token.
T12	Token selling	The IdP token can be easily transferred to a vote buyer. Note that this is an issue with many remote voting systems, including postal vote.	Removed use of the token.
T15	IdP & AP collusion	Collusion of the IdP and AP would allow linking a ballot to an identity.	Removed AP.
T17	AP forges identities	A malicious AP could fund wallets of arbitrary users, allowing them to bypass authentication.	Removed AP.
T18	AP voting	A malicious AP could create and fund its own wallets, allowing it to bypass authentication.	Removed AP.

Table 8.2: Mitigated threats

8.1.1 Ballot-Secrecy

Thanks to the DKG, the secrecy of a ballot (made public on the bulletin board) is provided under the assumption that at least one sealer remains honest – as all authorities are required in order to decrypt any value. The ballot ZKPs, do not leak any information, thus the ballot remain secret.

The secrecy of the ElGamal scheme has already been proven in previous work, and will not be further discussed here. However, the add-on – the **Randomizer** – warrants additional analysis. In order to achieve BS, we need to show that the randomizer does not learn the contents of any ballots it randomizes.

The randomizer receives the ElGamal encrypted ballot, two DCP proofs, one wFS, and one sFS. The secrecy of the encrypted ballot relies on the DDH as mentioned [13]. The DCP proofs are NIZKPs and thus leak no information under the random oracle assumption. The randomizer then simply computes an encryption of 0 and adds it to the original ballot. In order to do this, no further information is needed on the ballot. Thus, the randomizer learns nothing about the voter's choice.

8.1.2 Correctness

The correctness of the tally is given thanks to the use of DCP to prove the correct decryption of the values. The usage of the BC as a public bulletin board allows anyone, voters included, to verify the correctness of the execution.

A ballot will provably contain only 0 or 1, assuming that voter and randomizer do not collude. This is due to the fact that the re-randomized ballot will apply the *weak Fiat-Shamir* (wFS) transformation to yield a diverted proof for the ballot. The wFS heuristic has been shown [29] to be flawed, in that a prover can construct a proof that also verifies if the value does not lie in the correct range. Assuming either the voter or the randomizer remains honest, an incorrect ballot will not be recorded on the PBB.

If a *voter is malicious*, but the *randomizer is honest*: the voter must deliver a ballot proof using the strong Fiat-Shamir (sFS) transformation to the randomizer, which it verifies. Thus, the wFS cannot be used in this case.

If a *randomizer is malicious*, but the *voter is honest*: the randomizer must produce a proof of correct re-encryption using the sFS. The proof of re-encryption proves to the voter that the ballot still encrypts the original message m, as intended by the voter. So, the randomizer could not construct an invalid ballot and prove to the voter that the re-encryption still represents the intended message m, without the voter having noticed it.

As such, the only scenario in which an invalid ballot could be cast, is in case of collusion between a randomizer and a voter, which unfortunately raises the trust in the randomizer. Future work could try to derive a divertible proof applying the sFS heuristic instead of wFS.

8.1.3 Receipt-Freeness

To show that it is receipt-free, there are a couple of observations to highlight:

- A voter does not at any point learn the nonce used in the randomization. Due to this, she cannot reconstruct the ballot which goes on the PBB. Proofs that the randomizer produces are all ZKPs and, as such, do not leak the randomization witness: if the voter were to redo the process, the randomizer would use a new nonce – hence, the randomization of the ballot is not reproducible. Conversely, it is important to note that if the randomizer and the voter (or the vote buyer) were to collude, RF would not hold anymore. BS, however, remains intact.

- The voter receives a designated-verifier proof of correct re-encryption. This means that the proof is not transferable to a third-party for verification: if it were, a voter could send her ballot, the re-encryption proof, and the encrypted ballot on the PBB, and any third-party could verify the relation, breaking RF. The use of the designated-verifier proof means that the proof makes sense to Alice (the voter), but not to a vote seller.

- The diverted DCP proof of validity is chosen uniformly at random and is thus indistinguishable from the original proof. The initial version of the algorithm used for this thesis did not *fully* render the proof indistinguishable, and would thus leak information which would break RF. This was on account of the fact that the initial draft did not blind the commitments, but only the response-part of the DCP proof. In order to achieve RF, the voter must not be able to reproduce any piece of information on the PBB – this includes the proof. Simply blinding the ballot, but being able to reproduce the proof (which uses the same nonce, among other random parameters employed for the encryption), would indicate that the voter knows the nonce used in the encryption with non-negligible probability. Thus, it would be possible to determine a relation between the proof produced by a voter and her ballot, posted on the PBB. While it is not yet entirely proving *how* a voter voted, this might be enough to convince a vote buyer. For this reason, it is important to fully blind the proof which goes on the PBB.

8.1.4 Coercion-Resistance

The system does not provide Coercion-Resistance.

Swiss legislation does not permit to cast multiple ballots, which is often mentioned as a way to provide CR. However, since the voter's wallets would be known to coercers, multiple ballots from the same address would not protect a voter from repercussions, which is the main point of CR. Provotum 2.0 [7] also did not provide CR.

8.1.5 End-to-end Verifiability

In order for a system to fulfill End-to-end Verifiability, a system has to provide the following properties.

Cast-as-Intended

The system at present does not provide CaI. The only means of auditing the contents of a ballot would be to record the nonce employed in the encryption and use it to reconstruct the pre-randomization ballot. The recreated pre-randomization ballot, together with the proof of randomization, can be used to audit the contents of an encrypted ballot as published on the PBB. Recording the nonce, might increase the attack surface for an attacker to learn the contents of the voter's ballot.

Many systems advertise possessing CaI under the assumption of an honest client device. However, this seems unjustified in the context of remote voting. It is hard to have confidence that a voter's personal computer would be properly maintained, patched, and free of malware. Provotum 2.0 [7] was also missing CaI.

Recorded-as-Cast

Once the ballot and proof have been cast, these are stored directly on the BC without going through intermediaries. As such, a voter can easily verify that the ballot was recorded as cast, by checking if the generated ballot and proof match what is stored on the chain.

Counted-as-Recorded

With the information stored on the BC (*i.e.,* ballots and proofs), anyone, including the voters, can verify that the steps were computed correctly. Any attempts at manipulation would be immediately noticeable to any observer.

In conclusion, due to the missing Cast-as-Intended property, the system does not fulfill End-to-end Verifiability.

8.1.6 Eligibility Verifiability

The identity provider (IdP) is responsible of verifying voter-eligibility. The BC verifies that each ballot comes from an address, which was verified through blind signatures provided by an IdP.

Thanks to this mechanism, the IdP cannot trace an identity to a wallet address and to a ballot.

The IdP remains, however, a central weak-point in the scheme, as a rogue IdP could:

- sign its own wallets to participate in the vote, which could go undetected, as not all registered voters would vote.
- deny signatures to suppress voters, which would be noticeable.

Additionally, a mechanism based on blind signatures provides also concerns on the more practical side. In particular, the following two points are possible examples, both relating to the security of the voter's credentials, but handling two different aspects:

- If a voter's credentials were stolen and used to sign an address and cast a ballot: the voter would notice when requesting a signature, as the IdP would refuse, since it already signed a message for the same voter. The ballot, however, could not be invalidated, since no one, aside from the thief, would know the thief's address.
- Swiss law (VEleS, Art. 4) states: "*If the client-sided authentication measure is sent electronically, voters who have not cast their vote electronically must be able to request proof after the electronic voting system is closed and within the statutory appeal deadlines that the system has not registered any vote cast using their client-sided authentication measure.*" [66]. The system could provide proof that the credentials were used to register an address, as the IdP could store the blind signature and the identity of a voter. However, definitive proof of whether a vote was effectively cast would not be possible.

8.2 Scalability

In order to apply the system to real-world vote, it is necessary for the system to be able to scale. First, we draft the theoretical runtime from counting the number of modular exponentiations. Afterwards, we give a quantitative evaluation of the performance through benchmarks.

Modular exponentiations (`modExp`) are the most computationally intense steps of many cryptographic algorithm [67]. As such, we can often compare the performance of schemes based on the amount of `modExp` operations conducted. In particular, we are interested in the number of operations as a function of votes cast. With the help of Table 8.3, we estimate the increased computational cost of Provotum 3.0 in comparison to Provotum 2.0, for a single election with two choices. Provotum 2.0 had a total of 17 `modExp` per ballot, while the performance penalty of adding Receipt-Freeness is 26 additional `modExp` operations. The computation is divided between the randomizer (18 operations) and the voter (8 operations), with no additional overhead to the BC network.

In order to gauge the real-world performance of the algorithms, we run benchmarks of the voting algorithms. Ideally, the runtime of any protocol should grow linearly with the number of voters and votes cast. To get an idea of the order of magnitude of the tests, we can take a look at past votes in Switzerland to gauge voter participation. The popular vote of the 27th September 2020, had a voter turnout of around 3'000'000. Thus, the following evaluations takes 1'000'000 as a target value.

	Provotum 2.0		Provotum 3.0		
Step	Voter	BC	Voter	BC	Randomizer
Ballot Encryption	3		3		
DCP generation	6		6		
Re-Encr. proof generation					2
Re-Encr. proof verification			2		
DCP randomization			6		16
DCP verification		8		8	
Homomorphic sum					
Total by entity	9	8	17	8	18
Total	17		43		

Table 8.3: Number of `modExp` operations by entity, in order to cast a single "*yes*" or "*no*" vote.

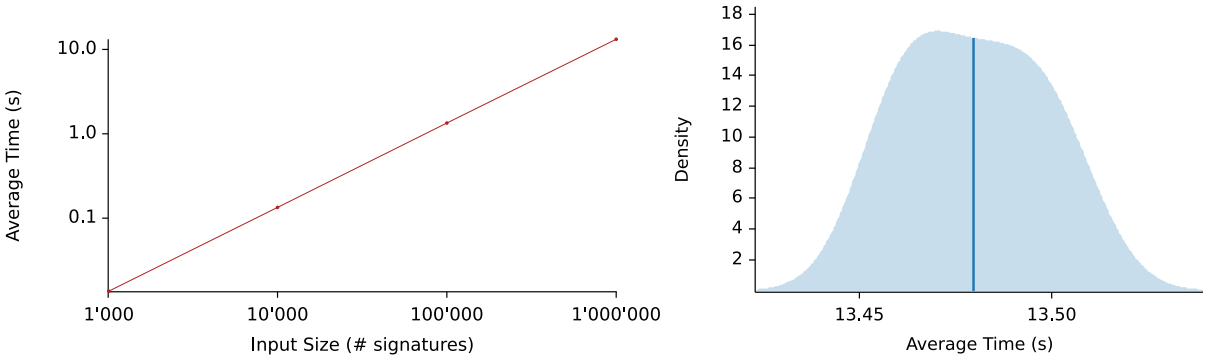
As mentioned in the security analysis in Chapter 5, to avoid availability issues, it would be best to run the on-chain steps off-chain and then to publish them. Thus, we evaluate directly the performance of the algorithms.

The benchmarks were computed by running the given algorithm on an otherwise idle machine. Each algorithm ran at least 10 times and up to 100 times, depending on the runtime of a single iteration. All of the line charts illustrated below use logarithmic scales on both axis – thus, a straight 45 degree slope represents a linear growth. The tests ran on a DigitalOcean server[1] *on server-grade hardware: a dedicated 32-core CPU Intel Xeon Gold 6140 CPU, running at 2.30 GHz per core, and 64GB RAM. The total runtime to compute all benchmarks was approximately 315 minutes, while the estimated monthly cost of operation is 640$.*

Firstly, the possible bottlenecks of the Provotum scheme have to be identified. These are protocols which require operating on many elements at the same time, or in a short period of time. The **pre-voting phase**, which involves only the set of sealers and the VA, does not need to scale, as this would involve few entities and is not time-critical. Then we have voter-registration. The BC is responsible of verifying the signatures of every eligible voter.

The benchmark in Figure 8.1a shows a very clear linear progression and places the runtime of verifying 10^6 signatures at an average of 13.48 seconds, with a throughput of 74.186 Kelements per seconds. Figure 8.1b shows the probability density function (PDF) of verifying 10^6 signatures over 10 iterations of the benchmark. The area under the curve reflects the probability of any given run to achieve a certain runtime.

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

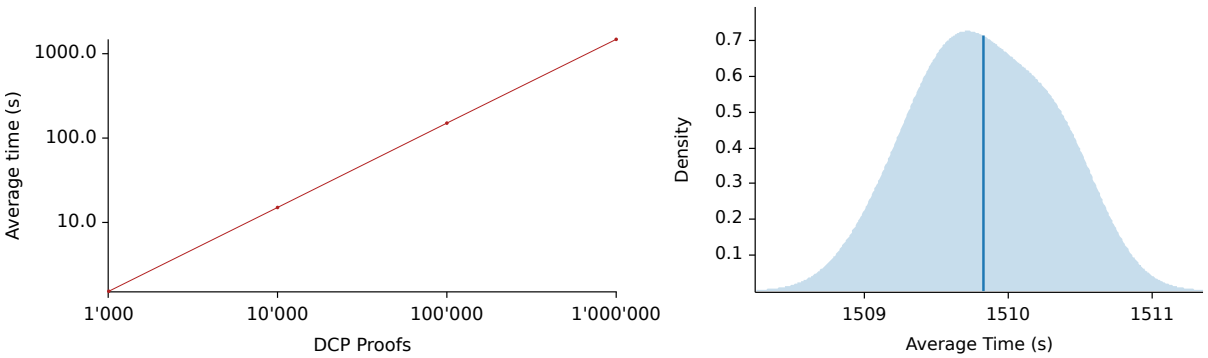


(a) Verification of 10^3 to 10^6 RSA signatures

(b) PDF over 10 iterations at 10^6 signatures to verify

Figure 8.1: Runtime for verifying RSA signatures

The **voting phase** is composed of *(i)* the ballot generation protocol on the voter's client device, *(ii)* the interactive randomization protocol, and *(iii)* the ballot verification protocol. Of the three, the first only runs once on each device. The second protocol is more involved and includes multiple messages sent between the voter and the randomizer. This step needs to scale to allow many voters to participate in the vote. However, this can scale easily by adding multiple load-balanced randomizer entities, or replacing the randomizer server with a hardware token, which would completely eliminate the scalability issue. The third step, is the biggest bottleneck, because the PoA BC can be seen as a single unit of computing: due to the Aura consensus algorithm, whoever turn it is when it receives a ballot, will need to verify it. The BC does not provide any benefit in terms of scalability, as it does not pool its computational resources. For this reason, we further analyse the runtime of the ballot validity proof (DCP) verification.



(a) Verification of 10^3 to 10^6 DCP proofs

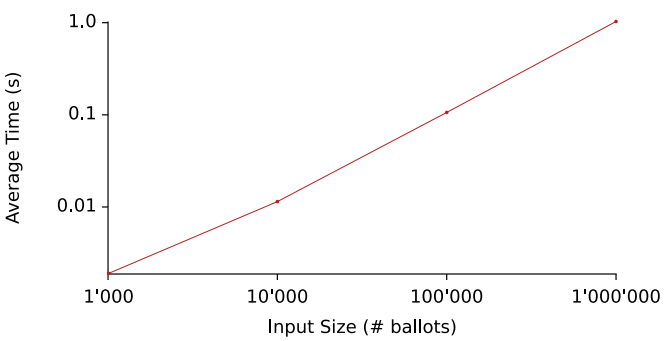
(b) PDF over 10 iterations at 10^6 DCP proofs to verify

Figure 8.2: Runtime for verifying DCP proofs

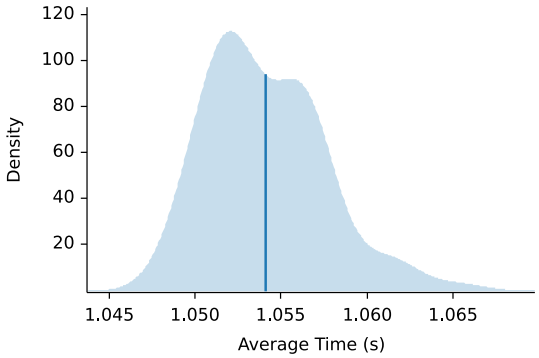
Figure 8.2 shows the runtime for verifying DCP proofs, for increasing amount of ballots. The benchmark cast 1'000, 10'000, 100'000, and 1'000'000 ballots to verify. The runtime demonstrates a linear growth, with a mean and median runtime of about 1510 seconds

and a throughput of 662 elements per seconds. This is the slowest algorithm to compute. This places it last in terms of speed among all protocols, which is to be expected as the verification of the DCP proofs requires the most modular exponentiations than any other algorithm (*i.e.,* 8 `modExp` per ballot).

Finally, we have the computation of the final tally in the **post-voting phase**. This is composed of the homomorphic sum and decryption of the resulting ciphertext.

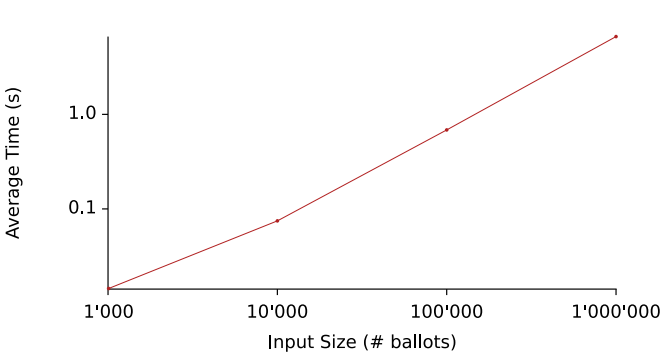


(a) Homomorphic sum of 10^3 to 10^6 ballots

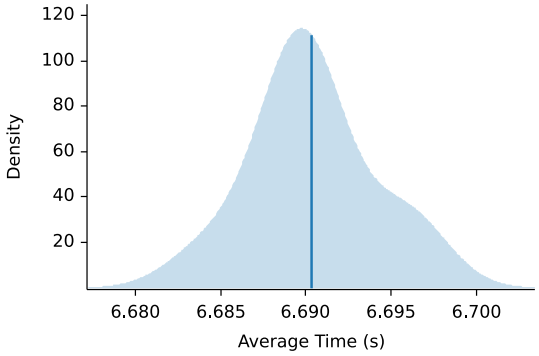


(b) PDF over 100 iterations at 10^6 ballots

Figure 8.3: Runtime for the homomorphic sum of ballots



(a) Decryption of the sum of 10^3 to 10^6 ballots



(b) PDF over 10 iterations of the decryption of 10^6 ballots

Figure 8.4: Runtime for decrypting the tally of ballots

Figure 8.3 and Figure 8.4 present the runtime of computing the homomorphic sum of up to 10^6 ballots and the final decryption of the tally. Again, they clearly demonstrate a linear growth with the number of ballots to operate on. It took an average of 1.05 seconds to sum up to 10^6 ballots and 6.7 seconds to decrypt the tally, with a throughput of 949 Kelements per second for the sum. As the decryption only operates on a single element (the ciphertext resulting from the sum of ballots), the throughput would not be meaningful in this context. Additionally, the time to decrypt the ballot is directly tied to the number m of "*yes*" votes, as the algorithm decrypts by brute-force searching for a value i, such that $g^i = g^m \pmod{p}$, for $i = 0, 1, \ldots, N$, where N is the total amount

of ballots cast, $m < N$, and g^m is the sum of all ballots. Taking again the results of the past votes as an example, the tally was set at 50%, *i.e.,* half of the ballots were "*no*" votes, and the other half were "*yes*" votes. So, the decryption of the sum of $1'000'000$ ballots is equivalent to computing $g^i = g^m \pmod{p}$ $500'000$ times. If all votes were "*yes*" votes, the decryption would take double the time, while if all were "*no*", it would complete immediately on the first attempt.

The benchmarks clearly show that while the system is scalable, the throughput of verifying the validity of the ballots is the bottleneck. One possible mitigation would be to separate ballots in three separate storage items: one for incoming, one for verified, and one for spoilt ballots. The system would then process pending ballots and put them in the appropriate storage after verification. The voter would then vote and check at a later time whether the ballot was confirmed as valid. This is, however, not unlike postal voting, where a voter casts her vote, but does not receive any immediate confirmation of whether the ballot is valid or not.

Chapter 9

Summary and Conclusions

This thesis set out with the goal of improving on the existing Provotum design and prototype [7]. One of the first step was to conduct a security analysis, in order to highlight attack vectors and limitations, which could be addressed in a new iteration of Provotum. This analysis brought to light a few new issues (Chapter 5), along some limitations previously known: the lack of Receipt-Freeness, the small key size due to the lack of support in the Ethereum VM, and issues with identity management, which would allow any user with tokens to vote, not only eligible voters.

With the discovered issues, a new version of Provotum was designed – called *ProvotumRF* – in a nod to BeleniosRF [49]. This new version brings Receipt-Freeness to the blockchain-based voting scheme Provotum. The prototype was rebuilt from the ground up using Substrate, to allow for arbitrarily large keys. The newly implemented prototype provides the same notions of ballot privacy and decentralization as its predecessor: *(i)* cryptographic material is generated on the voter's device, *(ii)* all proofs are verified on-chain in a fully transparent manner, and *(iii)* the distributed key generation scheme ensures that no single authority can break the secrecy of the vote. The new prototype then adds *(i)* Receipt-Freeness – in the form of a new entity called *randomizer* – and *(ii)* enforces voter-eligibility – through a blind-signature-based protocol. Finally, the use of Parity's Substrate provides the foundation for future work. An evaluation of the design in terms of the security properties defined in Chapter 3 and of the prototype in terms of scalability, concludes this thesis in Chapter 8.

9.1 Future Work

Any work worth mentioning should provide a starting point for future work, and ProvotumRF is no different. Inputs for future work range from usability to the never-ending open-problem of decentralized identity management.

9.1.1 Cast-as-Intended

Cast-as-Intended is provided only under the assumption that the voter has the cryptographic skills required to audit the ballot. Thus, future work would investigate approaches to be used to bring individual verifiability to the masses, in a decentralized setting. One such approach could be the use of dedicated, air-gapped smartphones, or devices similar to the ones used for online-banking services. Such systems could carry out the needed cryptographic operations, while being understandable to the layman. The federal ordinance VEleS [66] has a provision dedicated to this: "*If the voters use a special technical aid for verification, this must have been specifically developed for the secure storage of secret elements and for carrying out cryptographic operations, such as devices used for secure home-banking. In addition, the voters must be able to convince themselves of the fact that the aids operate correctly by casting test votes.*"

9.1.2 Multi-Way Elections, Limited Votes and Write-ins

Electronic voting is a complex field of research. So much so, that even the simplified model of a binary vote already brings considerable challenges to the table. The prototype does not provide any support for other types of vote. A simple scheme for limited votes, in which voter choose k out of N candidates, is provided in Hirt [33]. It proposes to encode the choices in a binary vector, where each element represents a candidate. All operations are then conducted element-wise, and a proof that the sum of the elements of the vector is in the range $[0, k]$ is provided in addition. Schemes for multi-way elections, where voters select a single candidate out of N, can either be modelled as a special case of k-out-of-N election. Write-ins are another type of election, commonly found in many countries, which is missing from Provotum. An interesting approach can be found in Kiayias and Yung [68]. It combines a mix-net approach with homomorphic encryption, in order to leverage the performance of homomorphic encryption with the flexibility of mix-nets. The possibility to cast spoilt ballots, usually done in sign of protest, is also missing.

9.1.3 Decentralized Identity Management

Privacy and eligibility verifiability remain in stark contrast to one another. While ProvotumRF improves the situation of identity management, the identity provider remains a trusted third-party system (TTP). As Nick Szabo stated: "*Trusted Third Parties Are Security Holes*" [69]. We simplify the model and reduce the scope of the work by assuming the existence of trusted components, but in essence we are shifting the responsibility of security around. Therefore, a system which does not rely on a TTP component for identity management should definitely be further investigated.

Bibliography

[1] Schweizerische Eidgenossenschaft Bundesamt für Komunikation. *Strategie " Digitale Schweiz "*. 2018. URL: https://strategy.digitaldialog.swiss/de/.

[2] Christian Killer and Burkhard Stiller. "The swiss postal voting process and its system and security analysis". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11759 LNCS. Springer, 2019, pp. 134–149.

[3] Staatssekretariat für Wirtschaft SECO. *National E-Government Study 2019*. Tech. rep. 2019. URL: https://www.egovernment.ch/en/dokumentation/nationale-e-government-studie-2019/.

[4] *E-Voting-Moratorium | Für eine sichere und vertrauenswürdige Demokratie*. URL: https://e-voting-moratorium.ch/.

[5] Raphael Matile and Christian Killer Zurich. "Privacy, Verifiability, and Auditability in Blockchain-based E-Voting". PhD thesis. 2018. URL: http://www.csg.uzh.ch/%20https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/mp-raphael-christian.pdf.

[6] Communication Systems Group. *E-Voting: Blockchain-based Remote Electronic Voting*. 2019. URL: http://www.csg.uzh.ch/csg/en/research/evoting.html.

[7] *Provotum - Blockchain-based Remote Electronic Voting*. URL: http://provotum.ch/.

[8] R Matile et al. "CaIV: Cast-as-Intended Verifiability in Blockchain-based Voting". In: *ieeexplore.ieee.org* (). URL: https://ieeexplore.ieee.org/abstract/document/8751413/?casa_token=QsihjyynLPcAAAAA:-uARPDLp-KZ_dC-GaJnjya_L-vVyGrxdn-FfBu22mcF0R_llwVWT_FwPz4RrBdDpKN0ch_DnKzg.

[9] Christian Killer et al. *Practical Introduction to Blockchain- based Remote Electronic Voting | Tutorial 2 | 2020 IEEE ICBC - IEEE International Conference on Blockchain and Cryptocurrency*. URL: https://icbc2020.ieee-icbc.org/tutorial-2.

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

[11] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the ACM* 21.2 (Feb. 1978), pp. 120–126.

[12] Whitfield Diffie and Martin E Hellman. "Multiuser cryptographic techniques". In: *Proceedings of the June 7-10, 1976, national computer conference and exposition*. 1976, pp. 109–112.

[13] Taher Elgamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[14] Cameron F Kerry and Patrick D Gallagher. *Digital Signature Standard (DSS)*. Tech. rep. NIST, 2013. URL: http://dx.doi.org/10.6028/NIST.FIPS.186-4.

[15] David Bernhard and Bogdan Warinschi. "Cryptographic voting — a gentle introduction". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8604 (2014), pp. 167–211. URL: https://link.springer.com/chapter/10.1007/978-3-319-10082-1_7.

[16] Dan Boneh. "The Decision Diffie-Hellman problem". In: *International Algorithmic Number Theory Symposium*. 1998, pp. 48–63. URL: http://link.springer.com/10.1007/BFb0054851.

[17] David Chaum. "Blind Signatures for Untraceable Payments". In: Plenum Press, 1983, pp. 199–203. URL: https://link.springer.com/chapter/10.1007/978-1-4757-0602-4_18.

[18] Pascal Paillier. "Public-key cryptosystems based on composite degree residuosity classes". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 1592. Springer Verlag, 1999, pp. 223–238. URL: https://link.springer.com/chapter/10.1007/3-540-48910-X_16.

[19] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. "A Secure and Optimally Efficient Multi-Authority Election Scheme". In: *European Transactions on Telecommunications* 8.5 (1997), pp. 481–490.

[20] Ben Adida. *Helios: Web-based Open-Audit Voting*. Tech. rep. 2008, p. 335.

[21] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. "Belenios: A Simple Private and Verifiable Electronic Voting System". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11565 LNCS. Springer Verlag, 2019, pp. 214–238. URL: https://doi.org/10.1007/978-3-030-19052-1_14.

[22] Mika Kojo and Tero Kivinen. *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*. RFC 3526. May 2003. URL: https://rfc-editor.org/rfc/rfc3526.txt.

[23] D Shanks. "Class number, a theory of factorization, and genera". In: 1971.

[24] Torben Pryds Pedersen. "A threshold cryptosystem without a trusted party". In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1991, pp. 522–526.

[25] Yvo Desmedt and Yair Frankel. "Threshold cryptosystems". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 435 LNCS. Springer Verlag, 1990, pp. 307–315.

[26] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "Knowledge complexity of interactive proof systems". In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208. URL: https://epubs.siam.org/page/terms.

[27] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. "Non-interactive zero-knowledge proof systems". In: *Lecture Notes in Computer Science (including sub-*

series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Vol. 293 LNCS. Springer Verlag, 1988, pp. 52–72.

[28] Amos Fiat and Adi Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In: *Conference on the theory and application of cryptographic techniques*. Springer. 1986, pp. 186–194.

[29] David Bernhard, Olivier Pereira, and Bogdan Warinschi. "How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2012, pp. 626–643.

[30] C. P. Schnorr. "Efficient signature generation by smart cards". In: *Journal of Cryptology* 4.3 (Jan. 1991), pp. 161–174.

[31] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. "Designated verifier proofs and their applications". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1996, pp. 143–154.

[32] Tatsuaki Okamoto and Kazuo Ohta. "Divertible zero knowledge interactive proofs and commutative random self-reducibility". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 434 LNCS. Springer Verlag, 1990, pp. 133–149.

[33] Martin Hirt. "Receipt-free K-out-of-L voting based on ElGamal encryption". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6000 LNCS. 2010, pp. 64–82.

[34] Yvo Desmedt and Pyrros Chaidos. "Applying divertibility to blind ballot copying in the Helios internet voting system". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 7459 LNCS. Springer. 2012, pp. 433–450.

[35] Hugo Jonker, Sjouke Mauw, and Jun Pang. "Privacy and verifiability in voting systems: Methods, developments and trends". In: *Computer Science Review* 10 (2013), pp. 1–30.

[36] U N General Assembly. "Universal declaration of human rights". In: *UN General Assembly* 302.2 (1948).

[37] Svetlana Z Lowry and Poorvi L Vora. "Desirable properties of voting systems". In: *NIST E2E workshop*. 2009.

[38] David L Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms". In: *Communications of the ACM* 24.2 (1981), pp. 84–90.

[39] Tal Moran and Moni Naor. "Receipt-free universally-verifiable voting with everlasting privacy". In: *Annual International Cryptology Conference*. Springer. 2006, pp. 373–392.

[40] Aggelos Kiayias and Moti Yung. "Self-tallying elections and perfect ballot secrecy". In: *International Workshop on Public Key Cryptography*. Springer. 2002, pp. 141–158.

[41] Philipp E Locher. *Unconditional privacy in remote electronic voting: theory and practice*. 2016.

[42] Josh Benaloh and Dwight Tuinstra. "Receipt-free secret-ballot elections". In: *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. 1994, pp. 544–553.

[43] Steve Kremer, Mark Ryan, and Ben Smyth. "Election verifiability in electronic voting protocols". In: *European Symposium on Research in Computer Security*. Springer. 2010, pp. 389–404.

[44] Benoˆ Chevallier-Mames et al. "On some incompatible properties of voting schemes". In: *Towards Trustworthy Elections*. Springer, 2010, pp. 191–199.

[45] Ronald L Rivest. "On the notion of 'software independence'in voting systems". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 366.1881 (2008), pp. 3759–3767.

[46] Wenbo Wang et al. "A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks". In: *IEEE Access* 7 (2019), pp. 22328–22370.

[47] Hugo Jonker and Jun Pang. *Bulletin Boards in Voting Systems: Modelling and Measuring Privacy*. Tech. rep. 2011. URL: https://ieeexplore.ieee.org/abstract/document/6045953/.

[48] Josh Benaloh. "Ballot Casting Assurance via Voter-Initiated Poll Station Auditing." In: *EVT* 7 (2007), p. 14.

[49] Pyrros Chaidos et al. "BeleniosRF: A non-interactive receipt-free electronic voting scheme". In: *Proceedings of the ACM Conference on Computer and Communications Security* 24-28-Octo (2016), pp. 1614–1625.

[50] Véronique Cortier and Ben Smyth. "Attacking and fixing Helios: An analysis of ballot secrecy". In: *Journal of Computer Security* 21.1 (2013), pp. 89–148.

[51] Y. Liu and Q. Wang. "An E-voting Protocol Based on Blockchain". In: (2017).

[52] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. "A smart contract for boardroom voting with maximum voter privacy". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10322 LNCS. Springer Verlag, 2017, pp. 357–375. URL: https://link.springer.com/chapter/10.1007/978-3-319-70972-7_20.

[53] Karola Marky et al. "What did I really vote for? On the usability of verifiable e-voting schemes". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 2018, pp. 1–13.

[54] Sandra Guasch and Paz Morillo. "How to challenge and cast your e-vote". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2016, pp. 130–145.

[55] David Galindo, Sandra Guasch, and Jordi Puiggalí. "2015 neuchtel's cast-as-intended verification mechanism". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9269. Springer Verlag, 2015, pp. 3–18. URL: https://link.springer.com/chapter/10.1007/978-3-319-22270-7_1.

[56] Uwe Serdult et al. "Fifteen years of internet voting in Switzerland: History, Governance and Use". In: *2015 2nd International Conference on eDemocracy and eGovernment, ICEDEG 2015*. Institute of Electrical and Electronics Engineers Inc., May 2015, pp. 126–132.

[57] Véronique Cortier and Cyrille Wiedling. "A formal analysis of the Norwegian e-voting protocol". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 7215 LNCS. Springer, Berlin, Heidelberg, 2012, pp. 109–128. URL: https://link.springer.com/chapter/10.1007/978-3-642-28641-4_7.

[58] Martin Hirt and Kazue Sako. "Efficient receipt-free voting based on homomorphic encryption". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 1807. Springer Verlag, 2000, pp. 539–556.

[59] Kazue Sako and Joe Kilian. "Receipt-free mix-type voting scheme - A practical solution to the implementation of a voting booth-". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 921. Springer Verlag, 1995, pp. 393–403. URL: https://link.springer.com/chapter/10.1007/3-540-49264-X_32.

[60] Byoungcheon Lee Kwangjo Kim. "Receipt-free electronic voting through collaboration of voter and honest verifier". In: (2000).

[61] Olivier Baudron et al. *Practical Multi-Candidate Election System*. Tech. rep. 2001.

[62] Bin Yu et al. "Platform-independent secure blockchain-based voting system". In: *International Conference on Information Security*. Springer. 2018, pp. 369–386.

[63] Moritz Eck, Alex Scheitlin, and Nik Zaugg. "Design and Implementation of Blockchain-based E-Voting". PhD thesis. 2020, p. 61. URL: http://www.csg.uzh.ch/.

[64] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. "The Attack of the Clones Against Proof-of-Authority". In: *arXiv* (Feb. 2019). URL: http://arxiv.org/abs/1902.10244.

[65] I. Fette et al. *RFC 6455 - The WebSocket Protocol*. Tech. rep. 2011. URL: https://tools.ietf.org/html/rfc6455.

[66] The Swiss Federal Chancellery (FCh). *Federal Chancellery Ordinance on Electronic Voting (VEleS)*. 2013. URL: https://www.admin.ch/opc/en/classified-compilation/20132343/index.html.

[67] Rolf Haenni, Philipp Locher, and Nicolas Gailly. "Improving the Performance of Cryptographic Voting Protocols". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11599 LNCS. Springer, 2020, pp. 272–288.

[68] Aggelos Kiayias and Moti Yung. "The vector-ballot e-voting approach". In: *International Conference on Financial Cryptography*. Springer. 2004, pp. 72–89.

[69] Nick Szabo. *Trusted Third Parties Are Security Holes*. 2001. URL: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/ttps.html.

Abbreviations

sFS	strong Fiat-Shamir
wFS	weak Fiat-Shamir

AP	Access Provider
API	Application Programming Interface
BC	Blockchain
BS	Ballot-Secrecy
CaI	Cast-as-Intended
CaR	Counted-as-Recorded
CR	Coercion-Resistance
DCP	Disjunctive Chaum-Pedersen
DDH	Decisional Diffie-Hellmann
DKG	Distributed Key Generation
ETH	Ethereum
E2E-V	End-to-End Verifiability
HTTP	Hypertext Transfer Protocol
IdP	Identity Provider
IND-CPA	INDistinguishability under Chosen-Plaintext Attack
I/O	Input/Output
IV	Individual Verifiability
MITM	Man-In-The-Middle
NGO	Non-Governmental Organization
NIZKP	Non-Interactive Zero-Knowledge Proof
PoA	Proof-of-Authority
PoS	Proof-of-Stake
PoW	Proof-of-Work
PBB	Public Bulletin Board
PBFT	Practical Byzantine Fault Tolerance
RaC	Recorded-as-Cast
REV	Remote Electronic Voting
RF	Receipt-Freeness
RFC	Request For Comments
RPV	Remote Postal Voting
RSA	Rivest-Shamir-Adleman
SC	Smart Contract
TTP	Trusted Third-Party

UP	Unconditional Privacy
UV	Universal Verifiability
VA	Voting Authority
VEleS	Verordnung der BK vom 13. Dezember 2013 über die elektronische Stimmabgabe
VM	Virtual Machine
Wasm	WebAssembly
ZKP	Zero-Knowledge Proof

Glossary

Access Provider (AP) An Access Provider is a first-party system acting on behalf of the voting authority, as a gatekeeper.

Authorization Authorization is the decision whether an entity is allowed to perform a particular action or not, e.g. whether a user is allowed to attach to a network or not.

AURA PoA round-robin consensus algorithm. Each sealer takes turn to produce a block.

Ballot A method to achieve secret voting. Some container for a vote which is kept secret.

Blockchain An append-only, immutable, distributed database, managed by a peer-to-peer network.

Ballot-Secrecy Property of a REV system which does not disclose a voter's vote.

Cast-as-Intended Property of a REV system which allows a voter to audit the contents of her own ballot, as a mitigation against malicious voting devices.

Identity Provider (IdP) An Identity Provider is a trusted third-party that verifies the identity and eligibility of a voter.

Distributed Key Generation (DKG) Process by which multiple entities collaborate to create a key pair.

Fiat-Shamir heuristic Process through which a ZKP can be turned into a NIZKP by replacing the verifier's random challenge with a cryptographic hash of the protocol's transcript.

Public Bulletin Board (PBB) A PBB is a component which provides an authenticated channel, providing transparency and verifiability to many schemes. It's main properties are [47]:

- Information can only be appended, not removed or modified.
- The board is public, *i.e.,* anyone may read the board.
- The board provides a consistent view to any viewer.

Proof-of-Authority (PoA) Consensus algorithm where a set of known authorities take turns in producing blocks. Identity at stake.

Proof-of-Stake (PoS) Consensus algorithm where the ability to produce blocks is tied to the influence of an entity in the network. Generally, tied to wealth.

Proof-of-Work (PoW) Consensus algorithm which requires to prove that a certain amount of computation effort took place in order to include a block in the chain.

Receipt-Freeness Property of a REV which does not allow a voter to prove how she voted.

Remote Electronic Voting (REV) A system to allow voting via internet.

Sealer A sealer is an entity running a node participating in the Proof-of-Authority blockchain, as a validator.

Smart Contract (SC) A piece of software deployed on a BC that triggers automatically based on some predefined input or condition.

Software Independence A voting system is software-independent if an undetected change or error in its software cannot cause an undetectable change or error in an election outcome.

Substrate BC development kit, developed by Parity.

Trusted third-party (TTP) An external component to a system meant to facilitate communication. It is assumed to behave correctly at all times.

Voting Authority (VA) A Voting Authority is an entity coordinating the vote. The VA is responsible for the BC bootstrapping process (with the Sealers), deploying the SCs, and opening and closing the vote.

Zero-knowledge Proof Mathematical process by which we prove a statement, usually knowledge of a secret, without the need to divulge it.

List of Figures

List of Tables

List of Listings

Appendix A

Installation Guidelines

The source code for ProvotumRF can be found at `https://github.com/provotum/ProvotumRF`.

To simplify running ProvotumRF a docker compose[1] manifest accompanies the project, ensure Docker version 19.03.13 and Docker Compose are installed 1.27.4 or higher. For local development, NodeJs[2] 12.18.03 and Rust[3] 1.46.0 or higher are required.

To run a demo of Provotum follow the instructions in the README of the infrastructure repository.

[1] `https://docs.docker.com/compose/`
[2] `https://nodejs.org/en/`
[3] `https://www.rust-lang.org/`

Appendix B

Contents of the CD

This thesis is accompanied by an archive of the following items:

- A PDF file of this report
- The LaTeXsource code of this report.
- The images of this report.
- The raw data from the benchmarks.
- The source code of the implemented prototype.
- The source code of the benchmarks.