Provotum: A Blockchain-based and End-to-end Verifiable Remote Electronic Voting System

Conference Paper · November 2020

CITATIONS
0

READS
57

8 authors, including:

Christian Killer
University of Zurich
11 PUBLICATIONS **9** CITATIONS
SEE PROFILE

Bruno Rodrigues
University of Zurich
41 PUBLICATIONS **390** CITATIONS
SEE PROFILE

Eder John Scheid
University of Zurich
34 PUBLICATIONS **102** CITATIONS
SEE PROFILE

Muriel Figueredo Franco
University of Zurich
31 PUBLICATIONS **64** CITATIONS
SEE PROFILE

Provotum: A Blockchain-based and End-to-end Verifiable Remote Electronic Voting System

Christian Killer, Bruno Rodrigues, Eder John Scheid,
Muriel Franco, Moritz Eck, Nik Zaugg, Alex Scheitlin, Burkhard Stiller
Communication Systems Group CSG, Department of Informatics IfI, University of Zurich UZH
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
[killer|rodrigues|scheid|franco|stiller]@ifi.uzh.ch, [moritz.eck|nik.zaugg|alex.scheitlin]@uzh.ch

***Abstract*—While the existence of Public Bulletin Boards (PBB) is often formulated as an assumption in related work on Remote Electronic Voting (REV) systems, this work here on Provotum focuses on the practical design and architecture of such a PBB, including its distributed execution. Further, Provotum leverages a public permissioned Blockchain (BC) as a PBB, where only authorized entities can sign blocks, while the general public can verify all BC data.**

Therefore, Provotum defines a new and fully decentralized BC-based REV system, which deploys a permissioned BC as a PBB and allows for the explicit distribution of trust across different permissioned BC nodes. Provotum is operated in a fully distributed fashion by using Smart Contracts (SC), Distributed Key Generation (DKG), Homomorphic Encryption (HE), and Cooperative Decryption (CD), as well as employing client-side encryption, which enables ballot secrecy, while the BC forms an audit trail, enabling public and End-to-end Verifiability (E2E-V).

I. Introduction

The digitization of governmental processes also includes a modernization of voting processes to diminish or replace traditional paper-based voting through Electronic Voting (EV) systems [33]. In general, the advantages of EV over paper-based systems are widely recognized [23], providing increased efficiency and transparency for the electoral process. However, there are plenty of non-technical aspects of each country's culture that impact how EV systems are implemented. Hence, EV systems can be broadly classified as those *(i)* including vote casting in a Remote EV (REV) setting, such as the ones implemented in countries like Switzerland and Estonia, or *(ii)* requiring in-person voting, for instance by visiting a ballot booth, as in EV systems in Brazil and India.

While EV and REV systems expose trade-offs between election verifiability and privacy, REV systems involve higher complexity, since they run outside a controlled environment, *i.e.,* outside of a prepared voting booth where eligible voters cast their votes. In general, REV introduces risks concerning voter coercion, forced abstention, technical risks regarding threats to the personal voting device (*e.g.,* browser vulnerabilities), and the security of the network link to the REV environment. Increasing verifiability means introducing transparency into the electoral process, which means that there is a higher risk to voter's privacy. In this sense, both EV and REV systems must seek to strike a balance between these properties that is accepted by each country's culture.

Voting systems always require verifiable evidence of a correctly executed voting process. An essential building block of any EV system is the Public Bulletin Board (PBB). PBBs need to serve as a publicly shared, append-only audit trail, offering a consistent view for verifiers [30]. Generally, EV schemes assume the existence of a secure PBB, often treating this assumption too casually for real-world scenarios, and hence practical deployment remains a challenge [25]. EV systems need to minimize the requirement for trust, too, *i.e.,* accomplishing weaker trust assumptions. Arguably, trust in authorities may always remain necessary, *e.g.,* for identity management (authenticating voters, checking eligibility) [10]. However, the execution of the voting protocol itself has to be fully transparent and verifiable. Remaining trust has be distributed among a set of trustees in EV systems, *e.g.,* by distributing private key shares among multiple authorities [12].

The emergence of public Blockchains (BC) highlighted novel ways to distribute trust across nodes [5]. Instead of relying on existing public permissionless BCs, the deployment of a public permissioned BC is an efficient way to distribute trust across permissioned nodes in a network [30]. Only authorities are authorized to sign blocks, while anyone can verify BC data and read from the BC, enabling public verifiability. Further, a BC-based architecture persists a global state in an immutable way, thus, allowing the PBB to achieve a fully distributed voting protocol with End-to-end Verifiability (E2E-V).

Distribution of trust is straightforward in paper-based voting systems, *e.g.,* the tallying of paper ballots can be easily split up across different locations. On the contrary, EV deployments generally lead to centralized architectures within a "single location", and thus, introduce a Single Point-of-Failure (SPoF), which is prone to attacks (*e.g.,* Distributed Denial-of-Service). Most EV systems based on Homomorphic Encryption (HE) distribute trust by using Threshold Cryptography to distribute private key shares among trustees. However, the distributed key generation is not done transparently, but among trusted servers, and without a publicly readable PBB [49].

Provotum's key goal is to distribute trust via the voting protocol and allow public verifiability through the deployment of a PBB as a public permissioned BC. Preliminary work on Provotum used a public permissioned Proof-of-Authority (PoA) Ethereum BC as a PBB [35]. Although this showcases end-to-end voting, the initial version relied on a centralized

server for encryption and proof verification, defeating the purpose of using a BC as a PBB. This deficit had been tackled with a new approach to Provotum here, still based on a public permissioned PoA of Ethereum, but the new addition is the voting protocol, which now offers fully distributed E2E-V.

Provotum offers *(i)* Distributed Key Generation (DKG), where keys for the election are generated in a distributed fashion [12], allowing for the distribution of trust and mitigating the risk of a SPoF. Furthermore, *(ii)* Cooperative Decryption (CD) is deployed: At the end of the voting process, the decrypted key shares need to be combined to decrypt the homomorphically tallied result, thus distributing trust in the decryption process. Additionally, *(iii)* E2E-V is provided, including a Cast-as-Intended, a Recorded-as-Cast, and Tallied-as-Recorded verifiability, mainly by using the public permissioned BC as a PBB [8]. Finally, *(iv)* Client-Side Encryption is used, which is a crucial part of any REV system since the encryption on the client-side requires careful considerations of the security of the browser [30]. This newly designed and prototypically evaluated E2E-V protocol does not yet offer any Coercion-Resistance or Receipt-Freeness.

The remainder of this paper is organized as follows. While relevant background and related work is presented in Section II, Section III outlines cryptographic primitives. Based on Section's IV details on Provotum's design, assumptions, and stakeholders, Section V defines Provotum's voting protocol. Built on Section's VI description of the prototype, Sections VII and VIII render the evaluation and discussion. Finally, Section IX draws conclusions and outlines future work.

II. Background and Related Work

Research on REV systems spans 40 years, refining theoretical properties to define private and verifiable REV systems [26], from which background and related work is reviewed.

A. Blockchains and Distributed Ledgers

A Blockchain (BC) is an immutable backward linked-list formed by blocks of transactions. The BC is maintained by a distributed network of peers following a consensus mechanism [5]. Permissionless BCs represent the majority of current BCs, such as Bitcoin [38] and Ethereum [57]. Permissionless BCs allow anyone to join and leave the BC network at any time, as there is no central authority managing participants of the network. Thus, the BC is open to be written to and read by anyone as long the protocol is followed. In a permissioned BC, referred to as a Distributed Ledger (DL), a central authority or a consortium grants access rights to authorized parties. Further distinctions depend on whether a BC is public permissioned (*i.e.,* reading is possible for everyone) or private permissioned (*i.e.,* reading is restricted to selected entities).

BCs and DLs rely on consensus mechanisms to agree on the next state of the network and to maintain its consistency across all nodes. Traditional consensus mechanisms (*e.g.,* Byzantine Agreement Protocols [45]) are not suitable for BCs, as they are susceptible to Sybil Attacks [41]. Hence, a myriad of mechanisms, including Proof-of-Work (PoW) and Proof-of-Authority (PoA), were proposed to address specific BC and DL types [56]. PoW is mostly used by permissionless BCs, where miners solve a difficult mathematical problem, which determines whether they can append blocks or not. Such difficulty is adjusted depending on the network's computational power to ensure that blocks are produced in a fixed time interval. For a permissioned BC, PoA is suitable, since it does not depend on nodes solving mathematical problems, but instead uses a set of *authorities*, *i.e.,* nodes that are explicitly allowed to create new blocks. These authorities are usually referred to as *Sealers* or *Validators* of the BC. The new state of the BC has to be signed off on by the majority of authorities, in which case it becomes a part of the permanent DL in the form of an additional block. This setup has the advantage that it is less computationally intensive compared to the PoW mechanism and blocks can be created more predictably (*i.e.,* in more steady time intervals).

B. Related Work

Informally, privacy is defined by Ballot Secrecy (BS), Receipt-freeness (RF), and Coercion-resistance (CR) [11]. BS and privacy are synonymous in REV literature, but the term BS is favored to avoid confusion with other notions of privacy [54]. Thus, BS is concerned with the privacy of the ballot, whereas RF ensures that a voter cannot gain a receipt of how she voted to avoid vote selling and buying [4]. A scheme offers CR if it is infeasible for the coercer to determine if a coerced voter complies with the demands [28]. And perfect BS ensures that knowledge about the partial tally of the ballots of any set of voters is only computable by the coalition of all the remaining voters (this property captures strong voter privacy as understood in real-world elections) [29].

1) Properties: Three properties define verifiability in the context of EV [11]: *(i)* Individual Verifiability ensures that a voter can verify that the PBB includes the vote cast [28]. *(ii)* Universal Verifiability means that anyone can check that a PBB contains all votes of the final tally. *(iii)* Eligibility verifiability means that anyone can check that each ballot on the PBB was cast by a registered, eligible voter. At most, one ballot is tallied per voter [32]. Refinements of Eligibility Verifiability to Unforgeability ensure that only eligible voters can construct authorized ballots [54]. Furthermore, E2E-V consists of (i) Cast-as-Intended, (ii) Recorded-as-Cast, and (iii) Tallied-as-Recorded Verifiability [3]. However, "If voters and election observers do not have complete freedom regarding which software to trust to do the verification, or if only a privileged few can verify at all, the system is not truly E2E-V" [3].

Additional properties include the notion of Software Independence (SWI). The problem that SWI addresses is the difficulties in verifying correct casting, recording, and tallying done by sophisticated software in EV systems. Convincing anyone of the fact that the voting software is correct is arguably impossible, especially in a complex and full EV system. Furthermore, it is argued that the growing complexity and contradictory requirements lead to even more complex and

hard-to-test EV systems. Hence, the maxim followed by SWI is *Verify the election results, not the voting system.* [48].

2) REV Systems: BCs and DLs offer desirable properties for REV systems and are used as PBB's. BC-based REV often relies on a public permissionless BC as a PBB to store encrypted ballots and to provide verifiability of the final tally. *E.g.*, [34] relies on trust between a voter, a central organizer, and an inspector. The voter encrypts the vote with the central organizer's public key before the inspector signs it to cast a ballot. This system assumes that both actors can be trusted.

Hardwick et al. [24] proposes a system with a private Ethereum instance relying on a central certificate authority to authenticate voters; violating the ballot secrecy should the authority become byzantine. A variant of the Open Vote Network (OVN) [36] offers self-tallying of votes. All votes are publicly available once cast and the final tally can be verified by anyone. This leads to the drawback of fairness, since the last voter can tally the final result before anyone else can. OVN is based on the Ethereum BC and is practical for ≈40 voters. Seifelnasr et al. [52] proposes improvements to OVN, introducing an off-chain untrusted administrator to perform the bulk of the computations.

Similarly, Caforio et al. [7] proposes a decentralized voting system based on a mechanism that resembles a BC [40]. The entire voting system is based on a state machine that maps different stages of an election. For each stage, different protocols are used with the main premise of maintaining the privacy of votes, including their verification. The system relies on the ElGamal cryptosystem in combination with Neff Shuffles [39] to achieve privacy, and Non-Interactive Zero-Knowledge Proofs [50] for verifiability.

III. Cryptographic Instruments in Use

Cryptographic instruments are the key to building secure REV systems. Table I shows these instruments used with a concise mapping to Provotum's voting protocol phases and steps. Thus, a feasible approach to build REV systems is to employ Public Key Cryptography (PKC) based on Finite Fields or Elliptic Curve Cryptography (ECC). In combination with threshold cryptography, Homomorphic Encryption (HE) [50], Distributed Key Generation (DKG) and Cooperative Decryption (CD), private keying material can be distributed among trustees. Non-Interactive Zero-Knowledge Proofs (NIZKP) are used to prove the validity of ciphertexts.

A. Homomorphic Encryption (HE)

HE allows operations on encrypted data. Therefore, confidential data remains private, even when an operation is performed in an untrusted environment, *e.g.,* a public cloud provider [1]. This property can be used to tally votes in a REV system. Thus, every eligible voter can encrypt his or her vote using the election public key, resulting in an encrypted vote. No individual votes need to be decrypted because these votes can be added together homomorphically [26].

TABLE I: Cryptographic Instruments Applied in Provotum

Phases	Steps	Approach	References
Pre Voting	Registration	PKC	Ethereum Wallet [6]
	Pairing	ECC	ECDSA [57]
	Key-generation	DKG, NIZKP	Robust Threshold ElGamal Cryptosystem [12], Schnorr-Proofs [51]
Voting	Voting	PKC, NIZKP	ElGamal Cryptosystem [17], Disjunctive Chaum-Pedersen Proof [2]
Post Voting	Tallying	CD, NIZKP	Robust Threshold ElGamal Cryptosystem [12], Chaum-Pedersen Proof [9]
	Result	ECC	ECDSA [57]

1) ElGamal Cryptosystem: The ElGamal scheme is a PKC system defined over cyclic groups G and is based on the difficulty of solving the discrete logarithm problem in G [17]. ElGamal provides indistinguishability under chosen-plaintext attack and is provably secure under the Decisional Diffie-Hellman (DDH) assumption [14]. Also, ElGamal is homomorphic with respect to multiplication. Therefore, component-wise multiplication of two ciphertexts results in encryption of the product of respective plaintexts, which allows for the homomorphic tallying of encrypted votes without decrypting individual ballots [21]. ElGamal uses a cyclic group Z_p^*, a multiplicative group G^* defined over a finite field of integers Z_p (*i.e.,* multiplications of integers are always modulo p). For security reasons, the finite field with generator g and order of the group q are required to be a prime field, p and q are co-prime, and p should be at least 2,048 bits long.

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

$$(sk, pk) = (r, h) = (r, g^r \pmod p) \tag{1}$$

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

$$(c_1, c_2) = (g^r \pmod p, h^r m \pmod p),\ m \in \{0, 1\} \tag{2}$$

c) Decryption: To decrypt a ciphertext (c_1, c_2), the multiplicative inverse of the first component c_1 to the power of the private key sk is multiplied with the cipher's second component c_2 modulo p (3).

$$m = (c_1^{sk})^{-1} c_2 \pmod p \tag{3}$$

B. Distributed Key Generation

DKG and CD distribute trust and mitigate the risk of one party decrypting individual ballots. A threshold cryptosystem is used, where n out of n parties need to be present to decrypt a ciphertext [12]. This can be used in conjunction with the ElGamal public-key cryptosystem. The encryption algorithm for voters remains the same. However, the DKG and decryption steps need to be adjusted as follows.

The keypair is generated similarly to the standard ElGamal Key Generation. Each party $i \in [1, n]$ creates a key pair (sk_i, pk_i) (*cf.* Equation 4) resulting in n keypairs. While private keys sk_i are kept secret, public keys pk_i are collected and combined to form the system's public key $pk = h$ (*cf.* Equation 5), which will be used to encrypt these votes.

$$(sk_i, pk_i) = (r_i, h_i) = (r_i, g_i^r \pmod p) \tag{4}$$

$$pk = h = \prod_{i=1}^{n} pk_i \pmod p \tag{5}$$

C. Cooperative Decryption

CD is performed in two steps. First, each party decrypts the ciphertext (c_1, c_2) by using their private key sk_i, producing a decrypted share d_i (*cf.* Equation 6). These shares do not reveal anything about the plaintext. Then, decrypted shares are collected and combined to reveal the plaintext (*cf.* Equation 7).

$$d_i = c_1^{sk_i} \pmod p \tag{6}$$

(7)

input	ciphertext $= (c_1, c_2)$
	dec. shares $= [d_1, ..., d_i, ..., d_n]$
decryption	$m = \frac{c_2}{\prod_{i=1}^{n} d_i} \pmod p$
	$= c_2 \times (\prod_{i=1}^{n} d_i)^{-1} \pmod p$

D. Zero-Knowledge Proof

A Zero-Knowledge Proof (ZKP) is a cryptographic method by which a prover P can prove to a verifier V that it knows about a secret s (*e.g.,* the content of a vote) without revealing anything about s apart from the fact that it knows s. The proof is called zero-knowledge because V learns nothing about s apart from the fact that P knows about s [26], [53]. In REV systems, ZKPs can be used to generate proofs for valid ballots, without revealing anything about the voter's choice. This ensures that only valid votes are considered in the final tally [26], [53]. Moreover, ZKPs can also be used to prove that the decryption was done using the private key associated with the public key previously used for encryption.

The issue with interactive ZKPs is that the communication between the P and V is time-consuming, costly, and often not feasible in practice. A popular alternative is non-interactive ZKPs (NIZKP), which do not require any interaction between P and V. The interaction (*i.e.,* the challenge chosen by the verifier) is replaced by a cryptographic hash function serving as a random oracle. This is also known as applying the Fiat-Shamir heuristic [20] to an interactive ZKP. The following proofs are ZKPs transformed to NIZKP using the Fiat-Shamir heuristic. Thus, they can be used in REV systems without the need for interaction to generate and validate the proofs.

E. Schnorr Proofs

To prove knowledge of an ElGamal private key r, belonging to a public key h (*i.e.,* $h = g^r$), the Schnorr Proof [51] is suitable. It is a proof of knowledge of a discrete logarithm of $r = log_g(g^r)$ [22].

1) Schnorr Proof Generation: The proof can be created as shown in Equation 8. The private/public key pair (r, h) is needed as input. Additionally, a secure random value of a in the range $[1, q-1]$ is needed. $q = \frac{p-1}{2}$ is the additive modulus of the cyclic group Z_q^+ over the field F_q (*i.e.,* the cyclic group in which all additive operations take place) and g is the group's generator. The id is the unique identifier of the prover. To generate the proof, first, the commitment b, the generator g to the power of a, is hashed together with the unique id and the public key h. This forms the challenge c that is the first part of the proof. The second part of the proof is the response, which is the sum of the random value a and the product of the challenge c and the private key r.

(8)

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

2) Schnorr Proof Verification: To verify the Schnorr proof (c, d), first, the commitment b' and challenge c' need to be recomputed, as shown in Equation 9. Then, the recomputed challenge c' needs to match the original challenge c. And the generator g to the power of the response d needs to be equal to the recomputed commitment b' multiplied by the public key h to the power of the original challenge c. If both checks are successful, the proof verification concludes successfully.

(9)

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

F. Chaum-Pedersen Proof

The Chaum-Pedersen proof [9] can be used to affirm the correct decryption of a cipher (c_1, c_2) using the private key r. It is a proof of discrete logarithm equality of $log_g(g^r) = log_h(h^r)$ and is similar to the Schnorr Proof [12].

1) Chaum-Pedersen Proof Generation: To generate the proof for a ciphertext (c_1, c_2) (*cf.* Equation 10), a secure random value x in the range $[1, q-1]$ is needed. The commitment consists of two parts, a and b, a is equal to the cipher's first component c_1 to the power of the random value x, and b is equal to the generator g to the power of x. The challenge c is computed by hashing the unique prover id, the cipher's components c_1 and c_2, and commitments components a and b using a cryptographic hash function. The final response consists of f and d. f is computed by adding the random value x to the product of the challenge c and the private key r, and d is the cipher's first component c_1 to the power of the private key r. The proof itself consists of the commitment (a, b) and the response (f, d).

(10)

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

2) Chaum-Pedersen Proof Verification: The respective verification consists of the commitment (a, b) and the response (f, d) for a ciphertext (c_1, c_2) (*cf.* Equation 11). The challenge c' is recomputed by hashing the prover's unique id together with the cipher (c_1, c_2) and the response (a, b) extracted from the proof. To verify the proof's validity, two checks are performed: *(i)* the cipher's first component c_1 to the power of the response's fist component f needs to be equal to the commitment's first component a multiplied with the response's second component d to the power of the recomputed challenge c' and *(ii)* Whether the generator g to the power of the response's fist component f is equal to the commitment's second component b multiplied with the public key h to the power of the recomputed challenge c' is checked.

(11)

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

3) Disjunctive Chaum-Pedersen Proof: The disjunctive Chaum-Pedersen proof is a modification of the Chaum-Pedersen proof introduced in [2]. It is used for the encryption and is also called OR-proof or membership proof since the vote contained in the ballot is proven to be either a no-vote (value 0, subsequently represented as v_0) or a yes-vote (value 1, subsequently represented as v_1) and guaranteed to be nothing else. The proof does this without revealing the vote's actual content; thus, it is used during vote encryption.

4) Disjunctive Chaum-Pedersen Proof Generation: The proof generation is composed of two parts. The yes-part and the no-part for which only one part ever exists, *i.e.,* the knowledge of the actual encrypted choice and, in both cases, the contrary part needs to be simulated as well. *E.g.*, if the ballot contains a *yes*-vote v_1, the *no*-vote v_0 part of the proof needs to be simulated. The proof is generated with Equation 12, which is an example for a *yes*-vote proof, whereas c_0, f_0, x are secure random values in the range $[1, q-1]$ and r is the random value used in the encryption of the vote.

(12)

$$\begin{aligned}
\textbf{input} \quad & \text{ciphertext} = (a, b) \\
\textbf{sim.}(v_0) \quad & (a_0, b_0) = (g^{f_0}/a^{c_0} \pmod p, h^{f_0}/b^{c_0} \pmod p) \\
\textbf{proof}(v_1) \quad & (a_1, b_1) = (g^x \pmod p, h^x \pmod p) \\
\textbf{challenge} \quad & c = hash(id, a, b, a_0, b_0, a_1, b_1) \pmod q \\
& c_1 = c - c_0 \pmod q \\
\textbf{response} \quad & f_1 = x + c_1 \times r \pmod q \\
\textbf{output} \quad & v_1 \text{ proof} = (a_0, b_0, a_1, b_1, c_0, c_1, f_0, f_1)
\end{aligned}$$

In the case of the *no*-vote proof (*cf.* Equation 13), instead of simulating c_0, f_0, the values c_1, f_1 (secure random values in the range $[1, q-1]$) are simulated Another slight change is that instead of using b in the simulation, b/g is used. Actually, in the v_1 proof, b/g^0 is taken, which equals b. In the v_0 proof, where the *yes*-vote (of value 1) is simulated, b/g^1 is taken [2].

(13)

$$\begin{aligned}
\textbf{input} \quad & \text{ciphertext} = (a, b) \\
\textbf{sim.}(v_1) \quad & (a_1, b_1) = (g^{f_1}/a^{c_1} \pmod p, \\
& \qquad h^{f_1}/(b/g)^{c_1} \pmod p) \\
\textbf{proof}(v_0) \quad & (a_0, b_0) = (g^x \pmod p, h^x \pmod p) \\
\textbf{challenge} \quad & c = hash(id, a, b, a_0, b_0, a_1, b_1) \pmod q \\
& c_0 = c - c_1 \pmod q \\
\textbf{response} \quad & f_0 = x + c_0 \times r \pmod q \\
\textbf{output} \quad & v_0 \text{ proof} = (a_0, b_0, a_1, b_1, c_0, c_1, f_0, f_1)
\end{aligned}$$

5) Disjunctive Chaum-Pedersen Proof Verification: Both v_1 and v_0 proofs are validated using the algorithm described in Equation 14. The ciphertext serves as input *(i)*, while the recomputed challenge *(ii)* is followed by a comparison of hashes *(iii)* and concludes with various checks executed to validate the proof *(iv)*.

(14)

$$\begin{aligned}
\textbf{\textit{(i)}} \quad & \text{ciphertext} = (a, b) \\
& \text{proof} = (a_0, b_0, a_1, b_1, c_0, c_1, f_0, f_1) \\
\textbf{\textit{(ii)}} \quad & c' = hash(id, a, b, a_0, b_0, a_1, b_1) \pmod q \\
\textbf{\textit{(iii)}} \quad & c_0 + c_1 \pmod q \stackrel{!}{=} c' \\
\textbf{\textit{(iv)}} \quad & g^{f_0} \pmod p \stackrel{!}{=} a_0 \times a^{c_0} \pmod p \\
& g^{f_1} \pmod p \stackrel{!}{=} a_1 \times a^{c_1} \pmod p \\
& h^{f_0} \pmod p \stackrel{!}{=} b_0 \times b^{c_0} \pmod p \\
& h^{f_1} \pmod p \stackrel{!}{=} b_1 \times (b/g)^{c_1} \pmod p
\end{aligned}$$

IV. Provotum Design

Provotum is designed as a REV system based on a public permissioned BC, which serves as a PBB but also enforces the correct execution of the voting protocol. The use of a BC achieves public verifiability through an immutable, redundant data store that is persisted across a distributed network of BC nodes. Further, the full state of the application and the application logic (*i.e.,* the voting protocol) is immutably stored and executed on by all BC nodes in the distributed BC network, achieving redundant operation. While the first version of Provotum [35] stored all votes on-chain, it relied on a central server to handle encryption and proof generation, which is obviously infeasible in a real-world scenario. Therefore, this section focuses on the new Provotum design [47], [46], which still leverages a public permissioned BC as PBB. However, the voting protocol is now fully executed on-chain in the ballot's Smart Contract (SC).

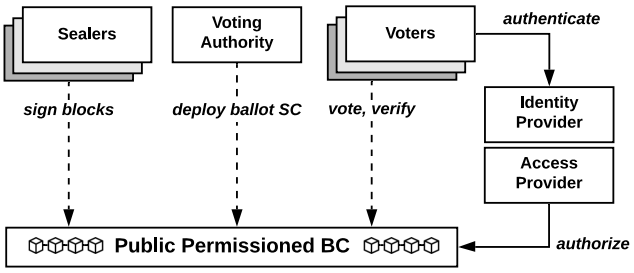

Fig. 1: Provotum System Design

A. Assumptions

The assumptions made for Provotum are congruent with [30] since the use-case of deployment is similar: deploying a REV system that takes the requirements of Switzerland, a federalistic state, into account. However, while [30] shows similarities in these assumptions, the voting protocol offers different properties, since DKG and CD are not implemented in the CaIV case.

TABLE II: Provotum's Design Assumptions as of [30]

ID	Description
$A1$	Every voter is only allowed to cast a single vote once.
$A2$	Every eligible voter is uniquely identifiable.
$A3$	The voting authority nodes are fixed and known *a priori*.
$A4$	The connection between the voter's voting device and the voting network is confidential.
$A5$	Submitted votes contain either 1 or 0, thus, votes are binary.

B. Public Bulletin Board

Key properties of a PBB are *(i)* readability by anyone, *(ii)* immutability of the information on the PBB, and *(iii)* the consistent state of the PBB for anyone viewing it [27]. Therefore, such a PBB is fundamental for any REV system, since auditable information is published during the execution of the voting protocol. Due to the Swiss federalistic structure, a public permissioned BC serves as an appropriate PBB without a central node, since this allows for the distributed execution of the voting protocol among trusted authorities [30]. Since all writers to the PBB are trusted authorities and known beforehand, there is no need to employ a PoW-based consensus mechanism to secure the network. Thus, a PoA consensus mechanism is a perfect fit given assumption A3 of Table II.

C. Stakeholders

1) Sealers: are authorities allowed to sign blocks and participate in the consensus algorithm. The advantage of using a BC over an always-online Trusted Third Party (TTP) is that weaker trust assumptions necessary because voters can publicly verify the BC's data without requiring access through any intermediaries. By doing so, there is no longer a SPoF. Furthermore, additional validators can participate in the PoA consensus (*e.g.,* non-governmental organizations or any other partially trusted organization) to further increase trust in the network [58]. Sealers participate in the DKG and also in the Cooperative Decryption during tallying, once the voting has ended. In the Swiss scenario, each canton can act as a trusted Sealer and operate such a service.

2) Voting Authority (VA): The VA is the voting administrator and acts mainly as a coordinator of the voting phase in Provotum. The VA is responsible *(i)* for coordinating the initial start-up phase of the BC with Sealers, *(ii)* deploying the voting SC on the BC and *(iii)* formally opening and closing the vote. In general, the definition of the VA depends on the vote at hand and could be handled by different organizations in collaboration (*e.g.,* the municipality, the canton, or the national government).

3) Voters: Voters are eligible citizens willing to participate in the voting process. The first voter's responsibility is to authenticate with the Identity Provider (IdP). The Access Provider (AP) will authorize the voter to cast his or her vote. Then, the voter can cast his / her vote and, if required, verify the E2E-V of the process by inspecting the PBB.

4) Identity and Access Provider: The IdP is a TTP responsible for the verification of voter eligibility. The IdP provides an authentication service that validates and verifies the Electronic Identity (E-Identity) of users trying to receive authorization for the REV system. After verifying the E-Identity, the IdP provides a one-time token that will be used by the voter to authorize with the AP. The AP is a service provided by the VA or another TTP. The AP provides an authorization service that only grants access to eligible voters by validating the voter's one-time token. If successful, the AP funds the voter's BC account with voting tokens (*e.g.,* in the case of a ballot SC requiring specific tokens), which allows the user to participate in the vote. Together, this provides secure and privacy-preserving authentication and authorization.

V. Provotum's Voting Protocol

The Provotum voting protocol is divided into three phases: *(1)* the Pre-Voting Phase, *(2)* the Voting Phase, and *(3)* the Post-Voting Phase (*cf.* Figure 2). The voting protocol is fully

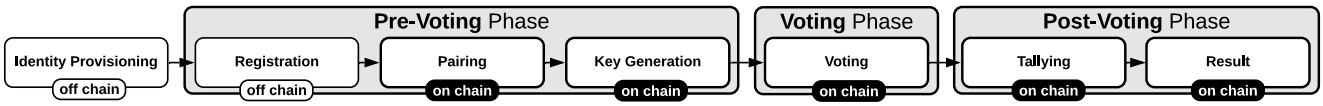

Fig. 2: Provotum Voting Protocol Phases, Sub-phases, and Overview

executed on the BC as soon as the ballot SC has been deployed during pairing, enforcing the distribution of trust among VAs.

A. Identity Provisioning

Identity Management (IdM) is dependent on the legal regulation and often formulated as an assumption in REV systems. However, to achieve an end-to-end voting process, Provotum includes a simplified IdM in order to evaluate the prototype. Before the voting protocol starts, an Electoral Register is created, containing eligible voters for the vote. The IdP generates a randomized one-time token for each eligible voter and sends a shuffled list of all tokens to the AP. The IdP verifies the identity of the voter (*i.e.,* a digital certificate issued by the government containing the unique ID of the voter) and checks if the voter is eligible for this vote. The voter (1) authenticates him-/herself with the IdP using his/her E-ID. If the eligibility verification is successful, (2) the IdP issues a one-time $token_v$ to the voter, then (3) the voter generates a new Ethereum account and submits it along with the $token_v$ to register with the AP. The AP verifies the one-time $token_v$, and (4) ensures that the voter's Ethereum account has sufficient funds to send a vote transaction. This step is only required because the prototype uses a permissioned BC based on Ethereum.

B. Pre-Voting Phase

The Pre-Voting phase sets up the Voting Phase in three steps.

1) Registration: During the registration, all Sealers generate public/private keys for the BC account and register the public key with the VA. Each cantonal Swiss Election Office (EO) can participate, allowing them to *seal* blocks in the BC.

2) Pairing: From the VA, all Sealers retrieve the BC genesis block, which defines a configuration file with addresses authorized to sign blocks (the Sealers). The VA acts as a bootstrapping node for the BC, helping all other Sealers to interconnect. Once all Sealers are sealing, the VA deploys the ballot SC to the BC. The ballot SC handles all vote-specific information (*e.g.,* the vote's system parameters, every encrypted vote, and proofs thereof). Now all subsequent actions are executed on all Sealers, distributing trust among them.

3) Key Generation: After the successful deployment of the ballot SCs, the DKG is initiated. Thus, all Sealers generate (a) an ElGamal public/private key pair and (b) a non-interactive Schnorr proof of knowledge, which are both sent to the ballot SC. This step ensures the distribution of trust among all participating Sealers, omitting any centralized SPOFs. Once every Sealer submits their public key and their proof has been verified, the VA initiates the generation of the public key for voting in the ballot SC, which will create the system's public key pk_{voting} by combining all the Sealer's public keys $pk_{vai}, i \in n$, where n is the total number of Sealers. The voting public key is used by all voters for vote encryption. Once the pk_{voting} has been generated, the VA can open the vote, which in most cases is a pre-defined and announced point in time.

C. Voting Phase

The Voting Phase includes three steps performed by the voter on his/her voting client: *(i)* the voter can proceed by making a voting decision and encrypting the vote. This is done automatically by the local voting client software, once the voter has selected his choice. In addition to the encrypted vote, the system will generate a NIZKP, proving that the encrypted vote contains one of the two possible answers (*i.e.,* yes or no) and nothing else without revealing the actual content of the vote; *(ii)* the encrypted vote and NIZKP are sent to the ballot SC, and vote casting is concluded as soon as the ballot SC first successfully verifies the proof and stores the encrypted vote; *(iii)* the voter will receive a vote confirmation transaction that can be used to check whether the vote had been Cast-as-Intended. Since all votes and proofs are stored on the BC, each voter can verify the inclusion in the ballot SC.

D. Post-Voting Phase

Once the voting period ends, which is a previously defined and publicly communicated point in time, the VA closes the vote by calling a function of the ballot SC and the Sealers can decrypt and tally the final result.

1) Tallying: The Sealers (i) retrieve all encrypted votes from the BC, (ii) add them homomorphically, perform partial decryption on a share of the final tally, and (iii) also generate a ZKP proving the correct decryption using the private key sk_{va_i} associated with the Sealer's public key pk_{va_i} stored in the ballot SC. The decrypted share and the proof are both submitted to the ballot SC, which verifies the validity of the proof, rejecting any invalid submissions. As a final step of tallying, any Sealer can trigger the final tallying simply by calling the tally function in the ballot SC, since all decrypted shares from the VAs are already persisted in the SC.

2) Result: All decrypted shares d_{va_i} combined allow for the generation of the decrypted result *i.e.,* the number of yes-votes submitted ($result = \sum_{i=0}^{n} d_{va_i}$). The ballot SC automatically subtracts the number of yes-votes from the total number of submitted votes in order to obtain the number of no-votes: $|votes_{no}| = |votes_{total}| - |votes_{yes}|$. The result is stored in the ballot SC and everyone can verify the proof of correct decryption or perform the decryption as well since all decrypted shares are stored in the ballot SC.

VI. Prototypical Implementation

Major technical details for the prototype are provided in detail in [47]. All cryptographic instruments are provided in a separate repository [46], allowing for modular re-use of the code. Provotum provides front-ends for all stakeholders. Thus, interacting with the REV system (and thus, the BC and PBB) does not require any additional software installation or low-level interactions because the system is fully controlled through browser-based dashboards. TypeScript [37] was used for all front and back-ends.

The back-end components (*e.g.,* the VA back-end, relaying interactions with the BC and the VA front-end) are also implemented in TypeScript and use Node.js [42]. Since the PBB is implemented as a public permissioned BC, Ethereum [19] was selected, since it offers Turing-complete SCs to distribute trust on-chain. For the Ethereum protocol implementation, the Parity Ethereum client (now called OpenEthereum) [43] client was used. All components are containerized using Docker [15] and can be started via a single script provided in [47].

A. Ballot Smart Contract (SC)

The ballot SC is the key, since it enforces the correct execution of the voting protocol. It contains the main application logic of Provotum and enables the distribution of trust by using SCs implemented in Solidity [18].

As indicated in Listing 1, the ballot SC enforces the voting protocol by using voting states, which are persisted and verified on-chain at the beginning of the function call (*cf.* Listing 4). By following a state-machine based approach, it is possible to ensure the correct execution order of the ballot SC and prevent actions, for example, performing the ”tallying” before the ”voting” actually happens.

```
enum VotingState{KEY_GENERATION, VOTING, TALLYING, RESULT}
```

Listing 1: Ballot.sol enum Voting States

The constructor of the ballot SC (*cf.* Listing 2) requires a voting question to be defined. Instead of directly storing a string, it could contain the cryptographic hash of a comprehensive official document, cryptographically signed and hashed by the authority (*e.g.,* an official voting document published by authorities on different online channels). The constructor requires the number of authority nodes to be defined, which is a numerical value for the authorities allowed to participate in the DKG. Thus, the distribution of trust is enforced from the beginning of the SC deployment.

```
constructor(string memory votingQuestion, uint256
    numberOfAuthNodes, address[] memory addresses) public
     {
        voteVerifier = new VoteProofVerifier();
        sumVerifier = new SumProofVerifier();
        keyGenProofVerifier = new KeyGenProofVerifier();
        election.votingQuestion = votingQuestion;
        priviledgedAddresses = addresses;
        NR_OF_AUTHORITY_NODES = numberOfAuthNodes;
    }
```

Listing 2: Ballot.sol Constructor

The constructor instantiates verifiers *(i)* to verify NIZKPs accompanied by public key shares submitted by Sealers during the DKG, *(ii)* to verify NIZKPs, proving the validity of ballots submitted and encrypted by voters, and *(iii)* to verify NIZKPs submitted with decrypted shares during final tallying. Privileged addresses define an array of addresses that belong to entities not allowed to vote (VA, Sealers, and AP).

Listing 3 shows an excerpt of the Election struct, which contains most of the variables relevant for the ballot SC, such as the voting question, the number of voters, and an array containing all voters, which, in turn, is a struct containing the triple of the voter's account address, the ciphertext encrypted with the election public key, and the NIZKP proving the validity of the encrypted ballot. The hasVoted mapping stores information on every address, including whether it had already cast a ballot, ensuring that no voter can vote multiple times.

```
struct Election {
    string votingQuestion;
    uint256 nrOfVoters;
    Voter[] voters;
    mapping(address => bool) hasVoted;
    mapping(address => PublicKeyShare) pubKeyShareMapping;
    address[] publicKeyShareWallet;
    mapping(address => DecryptedShare)
        decryptedShareMapping;
    address[] decryptedShareWallet;
    Cipher sumCipher;
    uint256 yesVotes;
    }
```

Listing 3: Ballot.sol Election Struct

Once the ballot SC is deployed, the DKG initiates the voting protocol on the BC. Each authorized entity is required to send a public key-share for the DKG. In Listing 4, each Sealer creates a transaction with a call to his/her submitPublicKeyShare function. This function requires *(i)* the voting state to be in KEY_GENERATION, *(ii)* the election public key to have not yet been set, *(iii)* the proof Schnorr NIZKP, which is verified in the KeyGenProofVerifier, and *(iv)* checking whether the Sealer has already submitted a key share. If all checks are successful, the key-share is persisted.

```
function submitPublicKeyShare(uint256 key,uint256 proof_c,
     uint256 proof_d) external
    returns (bool, string memory) {
    require(votingState == VotingState.KEY_GENERATION,
            'Need state KEY_GENERATION.');
    require(!IS_PUBKEY_SET,'Public key already set.');
    require(keyGenProofVerifier.verifyProof(proof_c,
        proof_d, key, msg.sender), 'Proof verification
        failed.');
    // check if this address submitted a share
    bool sealerAlreadySubmitted = false;
    for (uint256 i; i < election.publicKeyShareWallet.
        length; i++) {
        if (election.publicKeyShareWallet[i]==msg.sender){
            sealerAlreadySubmitted = true;
        }}
    PublicKeyShare memory publicKeyShare = PublicKeyShare(
        key, KeyShareProof(proof_c, proof_d));
    if (!sealerAlreadySubmitted) {
       // add sealer address to array
       election.publicKeyShareWallet.push(msg.sender); }
    // add or replace the share
    election.pubKeyShareMapping[msg.sender] =
        publicKeyShare;
    return (true, 'Proof verification succeeded.'); }
```

Listing 4: Ballot.sol Submitting Public Key Share for DKG

VII. Phase-based System Evaluation

Provotum's prototype has reached the stage of a fully working end-to-end voting process, which is evaluated in a Docker deployment. Following all steps provided in the voting protocol as above, including all stakeholders, the working prototype's evaluation follows the use-case of a complete end-to-end voting process for which videos are provided at [31].

The relevant stakeholders include the *(i)* Voting Authority (VA) in charge of the vote, *(ii)* Sealers authorized to sign blocks and participate in DKG and CD, and *(iii)* eligible Voters. It is assumed that voters have been authorized and identity provisioning was done a-priori (*cf.* Subsection V-A).

1) Registration Phase: This phase starts with each Sealer generating an Ethereum account. Each Sealer starts its back-end and front-end service, which will be served as depicted in Figure 3a. The front-end triggers the transmission of the public account address from the Sealer's back-end to the VA's back-end. The registration stage is concluded once all Sealers have sent their account address to the VA. The number of required Sealers can be pre-defined, or a time-limited registration could be opened beforehand. Once all Sealers have registered with the VA, the next stage is started by the VA.

2) Pairing Phase: During the pairing stage, the VA generates an initial configuration file (*i.e.,* genesis block) that all Sealers retrieve from the VA and then start their BC client, connecting to the bootnode, and bootstrapping the BC network to participate in the consensus. Even though this BC is permissioned, every transaction requires transaction fees. Hence, the configuration is used to ensure that all wallets of the VA, AP, and all Sealers show sufficient funds. The pairing phase concludes with the deployment of the ballot SC (which includes the definition of the voting question). The respective VA fron-tend is depicted in Figure 3b.

3) Key Generation Phase: During this phase, all Sealers generate an ElGamal public/private key pair (*cf.* Equation 1). Each Sealer generates a NIZKP, proving knowledge of the ElGamal private key belonging to the public key (*cf.* Equation 8). Once the keypair and the proof have been generated, the Sealer will send both the public key and the NIZKP directly to the ballot SC. The ballot SC first verifies the proof as shown in Equation 9 and only stores the Sealer's public key if the proof verification has been successful. Once every Sealer submits his/her public key and the respective proofs have been verified, the VA initiates the generation of the system's public key by triggering a function in the ballot SC. It will generate the system's public key pk_{system} by combining all Sealer's public keys $pk_{sealer_i}, i \in n$, where n is the total number of Sealers. Once pk_{system} is generated, the VA can open the vote. By opening the vote, the process advances to the voting phase.

4) Voting Phase: The voter front-end generates an Ethereum account and uses it together with the one-time token (provided by the Identity Provider) to authenticate with the AP. The AP verifies the one-time token and, if successful, funds the voter's account. The voter can now proceed by making a voting decision on the front-end, as depicted in Figure 3c. Once selected, the front-end encrypts the vote (*cf.* Equation 2). This is done automatically by the system once the voter has selected his choice. In addition to the encrypted vote, the system will generate the respective NIZKP (*cf.* Equation 12), proving the validity of the encrypted ballot. Once the vote has been encrypted and the proof has been generated, the voting client submits both directly to the ballot SC to avoid trusting any intermediary. The ballot SC first verifies the proof (*cf.* Equation 14) and, if successful, stores the encrypted vote. The respective voter's view is shown in Figure 3d, which allows for the verification on the BC with an integrated block explorer. Here, the dashboard shows a visualization of the BC's activity. Once the voting period has ended, which has to be previously defined and publicly communicated at some point in time, the voting authority will close the vote by calling a function from the ballot SC, concluding the voting stage.

5) Tallying Phase: During this phase, every Sealer fetches encrypted votes from the BC and homomorphically adds these encrypted votes to calculate the final sum. Afterward, each Sealer decrypts a share of the vote's final result and publishes it to the ballot SC. A NIZKP proof is generated as well (*cf.* Equation 10) to ensure the integrity of the decryption performed by the Sealer. The proof proves that the homomorphic sum has been decrypted using the private key sk_{sealer_i} associated with the Sealer's public key pk_{sealer_i} stored in the ballot SC. This ensures that only authentic decrypted shares are persisted, and someone apart from the owner of the private key sk_{sealer_i} (*i.e.,* the sealer) can submit a decrypted share for the respective public key pk_{sealer_i}. Once both the decrypted share and the proof have been generated, the Sealer will send both directly to the ballot SC. The ballot SC verifies the decryption proof as shown in Equation 11 and stores the decrypted share only if the proof verification was successful.

6) Results Phase: Once all Sealers have submitted the decrypted shares, and the ballot SC persists them, the VA triggers the final decryption step. The ballot SC combines the decrypted shares d_{sealer_i} and generates the decrypted result of all votes. The result is stored in the ballot SC, and anyone can verify the result by combining the decrypted shares.

VIII. Provotum's Analysis and Discussion

REV systems need to be analyzed, especially with respect to their privacy and verifiability, while considering additional properties, such as Software Independence (SWI).

A. Privacy

Privacy is crucial for REV systems, especially as a property of the voting protocol. Ballot Secrecy (BS) (also referred to as Ballot Privacy (BP)) requires that the content of a cast vote cannot be revealed under any circumstances and linked to a voter. Provotum guarantees Ballot Secrecy, since voters encrypt their votes using the system's public key (pk_{voting}), which was generated using each VA's public key share. However, no RF is provided yet, unless assuming a safe execution environment, running unchanged voting system code and not recording any of the cryptographic functions performed during

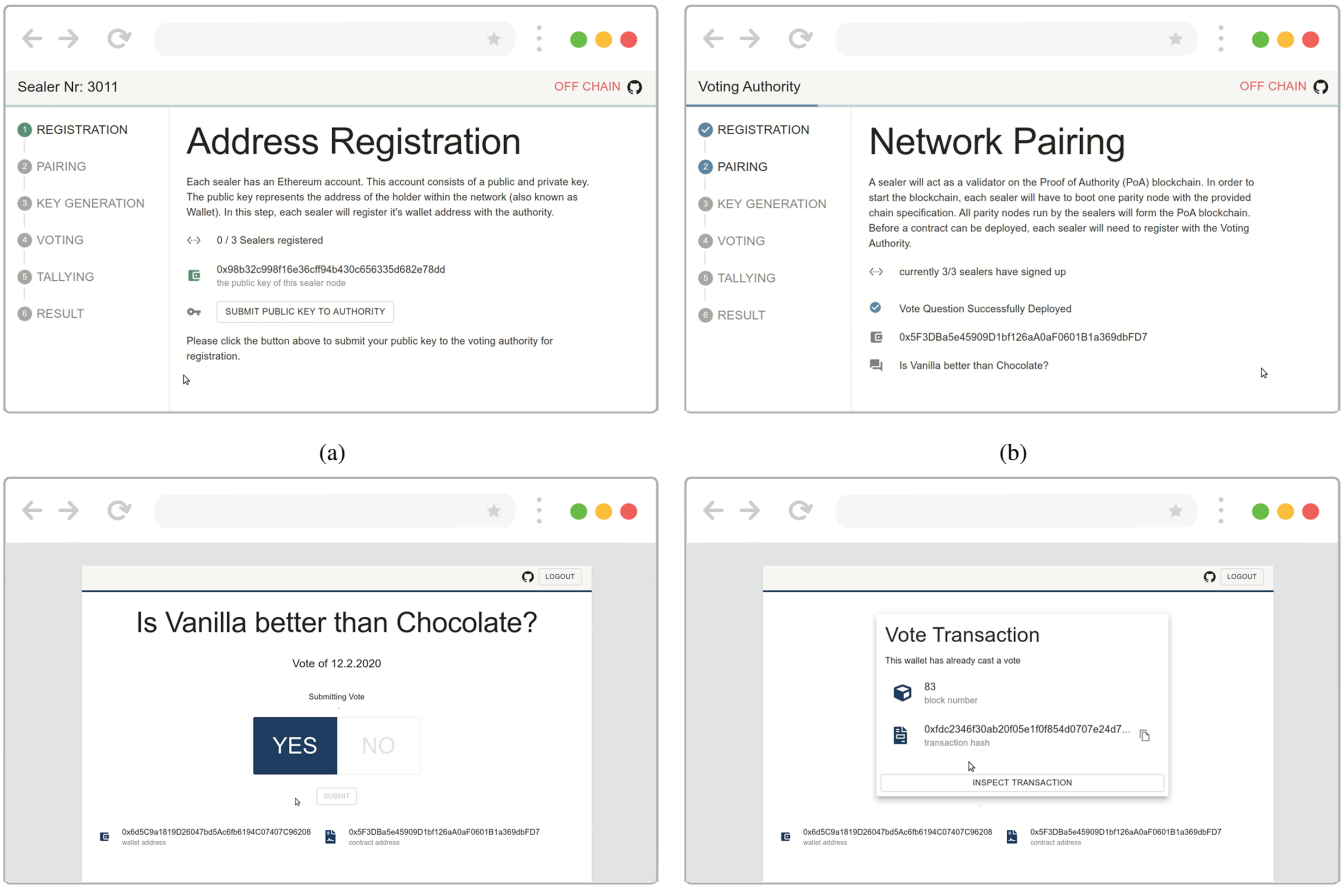

Fig. 3: Screenshots of Provotum's Frontend Prototype: (a) Sealer front-end during Registration Phase, (b) VA front-end during Pairing Phase, (c) Voter front-end during Vote Casting (Voting Phase), (d) Voter front-end after Vote Casting (Voting Phase)

execution. Therefore, RF (*i.e.,* the inability to prove to a third-party that a voter has cast a particular vote) is not yet given.

Although BS is obtained perfectly, CR cannot yet be guaranteed, which is a common problem in REV systems [26]. If a voter and coercer are at the same physical location, the voter can be forced to abstain or to give up his/her credentials. Since the voting process is observable in person, the voter can be observed making his/her voting choice. Also, Swiss law does not permit multiple vote submissions, *i.e.,* eligible voters are only allowed to cast their vote once [13]. In other countries, such as Estonia, voters are allowed to cast their votes multiple times, where only the last vote is counted toward the final tally [16]. Finally, long-term privacy (*e.g.,* everlasting privacy) cannot be guaranteed, since the Provotum protocol is based on the security of the ElGamal public-key cryptosystem, which is based on the DDH assumption.

B. Verifiability

Provotum provides E2E-V, since the voter can (a) check if his/her vote had been Cast-as-Intended (CAI), (b) if the BC included the vote Recorded-as-Cast (RAC), and (c) whether the votes had been Counted-as-Recorded (CAR). Since the voter encrypts a plaintext vote, generates a proof for its validity, and directly submits it to the ballot SC where the proof is verified and stored together with the encrypted ballot, the voter can check its inclusion into the ballot SC and thus, CAI verifiability is ensured once the ballot SC has verified the correctness of the proof. The voter will obtain a reference to his/her vote, *i.e.,* the transaction hash of the block, including his encrypted vote. The voter can retrieve the proof p and the encrypted vote v at any given time thanks to the reference (*i.e.,* the transaction hash of the block) obtained after submitting both to the BC. RAC verifiability is provided by comparing the locally-generated proof and the vote encrypted with the proof obtained from the BC. By recomputing the final decryption published by VAs, a voter can verify that CAR is provided.

C. Additional Considerations

The prototype[1] shows an implementation with easy-to-use characteristics and consistent dashboards. The current system does not yet scale to larger electorates. The main limitation is not due to Provotum, but due to the choice of the underlying BC as PBB. Thus, by replacing Ethereum as the implementation choice, the Provotum design with a public

[1]Provotum's source-code: http://provotum.ch

permissioned BC and its voting protocol is suitable to be implemented on top of another BC (*e.g.,* Substrate [44] or Hyperledger Fabric [55]). Furthermore, Weak SWI [48] is achieved in Provotum because anyone can fetch BC data and implement a verifier to check the proofs provided within the specification of the cryptographic instruments. A possible error could be detected since all BC data is immutably stored and distributed. However, if necessary, a recovery or roll-back will require additional mechanisms to be integrated.

IX. Summary, Conclusions, and Future Work

In summary, Provotum combines mathematically proven cryptographic primitives in a practical approach, prototyping all achievable properties, which can be analyzed within the BC-based REV system. Since BCs are distributed, immutable, and append-only, they are suitable for Public Bulletin Boards (PBB) in REV systems and Provotum contributes a working and fully documented proof-of-concept implementation. While previous work on Provotum employed a central server architecture [35], the new Provotum delivers a fully distributed system using a private permissioned BC as the PBB. This system practically enables the following major characteristics: *(i)* vote encryption and proof generation is handled on the voter's device, ensuring Ballot Secrecy; *(ii)* all cryptographic proofs are verified on-chain by the ballot SC instead of a central server; and *(iii)* the application of DKG and CD adds to the robustness of the system by distributing the trust any single node has to the entire network. The end-to-end voting process is available for demonstration purposes in videos at [31].

Concluding, the Provotum REV system is the first one in prototypical operation, offering full auditability of the entire end-to-end voting process deployed locally with pre-built Docker images [47], which are end-to-end verifiable, reaching Ballot Secrecy. While full code transparency is reached, Coersion-Resistance is not, since *i.e.,* re-submission of votes is not possible due to legal requirements in Switzerland.

The BC-based REV prototype still shows a few shortcomings. Provotum's properties hold for integers only up to a length of 256 bit, determined by the limit of the Ethereum Virtual Machine used. Furthermore, a full identity provisioning and management scheme would be used in practice to support the prototype. The scalability of cryptographic instruments deployed requires further performance evaluations. The lack of secure communication channels can be solved by deploying Onion routing or similar schemes. And Provotum does not yet support multi-way elections with a limited votes' support. Thus, future research will include additional protocol changes achieving RF, the employment of secure communication channels between the voter and the BC network, and dedicated encoding schemes for multi-way elections.

Acknowledgements

This paper was supported partially by *(a)* the University of Zürich UZH, Switzerland, and *(b)* the European Union's Horizon 2020 Research and Innovation Program under Grant Agreement No. 830927, the CONCORDIA project.

References

[1] F. Armknecht, C. Boyd, C. Carr, K. Gjosteen, A. Jaschke, C. A. Reuter, and M. Strand, "A Guide to Fully Homomorphic Encryption," *Cryptology ePrint Archive*, pp. 1–35, 2015. [Online]: https://eprint.iacr.org/2015/1192

[2] M. J. Atallah and M. Blanton, *Algorithms and Theory of Computation Handbook, Volume 1*, 2nd Edt. Chapman and Hall/CRC, 11 2009. [Online]: https://www.taylorfrancis.com/books/9781584888239

[3] J. Benaloh, R. Rivest, P. Y. A. Ryan, P. Stark, V. Teague, and P. Vora, "End-to-end Verifiability," *CoRR*, Vol. abs/1504.0, 4 2015. [Online]: http://arxiv.org/abs/1504.03778

[4] J. Benaloh and D. Tuinstra, "Receipt-free Secret-ballot Elections," in *26th Annual ACM Symposium on Theory of Computing (STOC '94)*. New York, New York, USA: ACM Press, 1994, pp. 544–553.

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

[6] V. Buterin, "A Next-Generation Smart Contract and Decentralized Application Platform," 2013. [Online]: https://ethereum.org/en/whitepaper/

[7] A. Caforio, L. Gasser, and P. Jovanovic, "A Decentralized and Distributed E-voting Scheme Based on Verifiable Cryptographic Shuffles," 2017. [Online]: https://www.epfl.ch/labs/dedis/wp-content/uploads/2020/01/report-2017-2-andrea_caforio-evoting.pdf

[8] D. Chaum, "Secret-Ballot Receipts: True Voter-Verifiable Elections," *IEEE Security & Privacy Magazine*, Vol. 2, No. 1, pp. 38–47, 1 2004. [Online]: http://ieeexplore.ieee.org/document/1264852/

[9] D. Chaum and T. P. Pedersen, "Wallet Databases with Observers," in *12th Annual Cryptology Conference (CRYPTO '92)*, Vol. 740 LNCS. Santa Barbara, California, USA: Springer, 1992, pp. 89–105.

[10] V. Cortier, D. Galindo, S. Glondu, and M. Izabachène, "Election Verifiability for Helios under Weaker Trust Assumptions," *Lecture Notes in Computer Science*, Vol. 8713 LNCS, No. PART 2, pp. 327–344, 2014.

[11] V. Cortier and B. Smyth, "Attacking and Fixing Helios: An Analysis of Ballot Secrecy," in *24th IEEE Computer Security Foundations Symposium (CSF '11)*, Abbaye des Vaux de Cernay, France, 6 2011, pp. 297–311.

[12] R. Cramer, R. Gennaro, and B. Schoenmakers, "A Secure and Optimally Efficient Multi-authority Election Scheme," in *17th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '97)*, Vol. 1233, Konstanz, Germany, 1997, pp. 103–118.

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

[14] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, 1976.

[15] Docker Inc., "Docker." [Online]: https://docker.com

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

[17] T. Elgamal, "A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms," *IEEE Transactions on Information Theory*, Vol. 31, No. 4, pp. 469–472, 7 1985.

[18] Ethereum, "Solidity," 2020. [Online]: https://solidity.readthedocs.io/

[19] Ethereum Foundation, "Ethereum." [Online]: https://ethereum.org/

[20] A. Fiat and A. Shamir, "How to Prove Yourself: Practical Solutions to Identification and Signature Problems," in *4th Advances in Cryptology Conference (CRYPTO '86)*, Vol. 263 LNCS. Santa Barbara, California, USA: Springer, 1987, pp. 186–194.

[21] R. Haenni, R. E. Koenig, P. Locher, and E. Dubuis, "CHVote System Specification," Cryptology ePrint Archive, Report 2017/325, 2017.

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

[23] F. Hao and P. Y. A. Ryan, "Real-World Electronic Voting: Design, Analysis and Deployment," *Real-World Electronic Voting: Design, Analysis and Deployment*, pp. 1–461, 2016.

[24] F. S. Hardwick, A. Gioulis, R. N. Akram, and K. Markantonakis, "E-Voting With Blockchain: An E-Voting Protocol with Decentralisation and Voter Privacy," in *IEEE International Conference on iThings, GreenCom, CPSCom and SmartData*. Los Alamitos, CA, USA: IEEE, 2018, pp. 1561–1567.

[25] L. Hirschi, L. Schmid, and D. Basin, "Fixing the Achilles Heel of E-Voting: The Bulletin Board," *IACR Cryptology ePrint Archive*, Vol. 2020, p. 109, 2020. [Online]: https://eprint.iacr.org/2020/109

[26] H. Jonker, S. Mauw, and J. Pang, "Privacy and Verifiability in Voting Systems: Methods, Developments and Trends," *Computer Science Review*, 2013.
[27] H. Jonker and J. Pang, "Bulletin Boards in Voting Systems: Modelling and Measuring Privacy," in *6th International Conference on Availability, Reliability and Security, (ARES '11)*, 2011, pp. 294–300.
[28] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-Resistant Electronic Elections," in *ACM Workshop on Privacy in the Electronic Society (WPES '05)*. Alexandria, VA, USA: ACM, 2005, p. 61–70. [Online]: https://doi.org/10.1145/1102199.1102213
[29] A. Kiayias and M. Yung, "Self-tallying Elections and Perfect Ballot Secrecy," in *6th International Workshop on Theory and Practice in Public Key Cryptography (PKC '02)*, D. Naccache and P. Paillier, Eds. Miami, FL, USA: Springer Berlin Heidelberg, 2002, pp. 141–158.
[30] C. Killer, B. Rodrigues, R. Matile, E. Scheid, and B. Stiller, "Design and Implementation of Cast-as-Intended Verifiability for a Blockchain-based Voting System," in *35th Annual ACM Symposium on Applied Computing (SAC '20)*. Brno, Czech Republic: ACM, 3 2020, pp. 286–293. [Online]: https://dl.acm.org/doi/10.1145/3341105.3373884
[31] C. Killer, B. Rodrigues, E. Scheid, M. Franco, and B. Stiller, "Practical Introduction to Blockchain-based Remote Electronic Voting," 2020. [Online]: https://github.com/christiankiller/icbc20-bcbev-tutorial/
[32] S. Kremer, M. Ryan, and B. Smyth, "Election Verifiability in Electronic Voting Protocols," in *15th European Symposium on Research in Computer Security (ESORICS '10)*, Vol. 6345 LNCS. Athens, Greece: Springer, Berlin, Heidelberg, 2010, pp. 389–404.
[33] R. Krimmer, "A Structure for New Voting Technologies: What They Are, How They Are Used and Why," in *The Art of Structuring: Bridging the Gap Between Information Systems Research and Practice*, K. Bergener, M. Räckers, and A. Stein, Eds. Cham: Springer International Publishing, 2019, pp. 421–426. [Online]: https://doi.org/10.1007/978-3-030-06234-7_39
[34] Y. Liu and Q. Wang, "An E-voting Protocol Based on Blockchain," *IACR Cryptology ePrint Archive*, p. 1043, 2017. [Online]: https://eprint.iacr.org/2017/1043.pdf
[35] R. Matile and C. Killer, "Privacy, Verifiability, and Auditability in Blockchain-based E-Voting," 2018. [Online]: https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/mp-raphael-christian.pdf
[36] P. McCorry, S. F. Shahandashti, and F. Hao, "A Smart Contract for Boardroom Voting with Maximum Voter Privacy," in *Lecture Notes in Computer Science*, Vol. 10322 LNCS. Berlin, Heidelberg: Springer, 2017, pp. 357–375.
[37] Microsoft, "TypeScript." [Online]: https://www.typescriptlang.org/
[38] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]: https://bitcoin.org/bitcoin.pdf
[39] C. A. Neff, "A Verifiable Secret Shuffle and Its Application to E-Voting," in *Proceedings of the 8th ACM Conference on Computer and Communications Security*, Ser. CCS '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 116–125. [Online]: https://doi.org/10.1145/501983.502000
[40] K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, I. Khoffi, J. Cappos, and B. Ford, "CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1271–1287.
[41] D. Ongaro and J. K. Ousterhout, "In Search of an Understandable Consensus Algorithm," *USENIX Annual Technical Conference*, pp. 305–319, 2014.
[42] OpenJS Foundation, "Node.js." [Online]: https://nodejs.org/en/
[43] Parity Technologies, "Open Ethereum." [Online]: https://github.com/openethereum/openethereum
[44] ——, "Substrate." [Online]: https://github.com/paritytech/substrate
[45] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *Journal of the ACM (JACM)*, Vol. 27, No. 2, pp. 228–234, 1980.
[46] Provotum, "Provotum 2.0 Crypto Library." [Online]: https://github.com/provotum/evote-crypto
[47] ——, "Provotum 2.0," 2020. [Online]: https://github.com/provotum/provotum-v2
[48] R. L. Rivest and M. Virza, "Software Independence Revisited," in *Real-World Electronic Voting: Design, Analysis and Deployment*, 2016.
[49] R&S Scytl, "Swiss Online Voting System Cryptographic proof of Individual Verifiability," 2017.
[50] K. Sako and J. Kilian, "Receipt-Free Mix-Type Voting Scheme," 1995, pp. 393–403.
[51] C.-P. Schnorr, "Efficient Signature Generation by Smart Cards," *Journal of Cryptology*, Vol. 4, No. 3, pp. 161–174, 1991.
[52] M. Seifelnasr, H. S. Galal, and A. M. Youssef, "Scalable Open-Vote Network on Ethereum," *IACR Cryptology ePrint Archive*, Vol. 2020, No. November 2019, p. 33, 2020. [Online]: https://eprint.iacr.org/2020/033
[53] W. D. Smith, "Cryptography Meets Voting," *Compute*, Vol. 10, p. 64, 2005.
[54] B. Smyth, "A Foundation for Secret, Verifiable Elections," *IACR Cryptology ePrint Archive*, p. 225, 2018. [Online]: http://eprint.iacr.org/2018/225
[55] The Linux Foundation, "Hyperledger Fabric." [Online]: https://github.com/hyperledger/fabric
[56] W. Wang, D. T. Hoang, P. Hu, Z. Xiong, D. Niyato, P. Wang, Y. Wen, and D. I. Kim, "A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks," in *IEEE Access*, Vol. 7, 1 2019, pp. 22 328–22 370.
[57] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger." *Ethereum Project Yellow Paper*, 2014.
[58] K. Wüst and A. Gervais, "Do you Need a Blockchain?" in *IEEE Crypto Valley Conference on Blockchain Technology (CVCBT '18)*. Zug, Switzerland: IEEE, 2018, pp. 45–54.

All links provided above were last accessed on October 15, 2020.