Design and Implementation of Cast-as-Intended Verifiability for a Blockchain-based Voting System

Christian Killer, Bruno Rodrigues, Raphael Matile, Eder Scheid, Burkhard Stiller
Communication Systems Group CSG, Department of Informatics IfI, University of Zurich UZH
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
[killer,rodrigues,scheid,stiller]@ifi.uzh.ch,raphael.matile@bluewin.ch

ABSTRACT

Digitization of electoral processes depends on confident systems that produce verifiable evidence. The design and implementation of voting systems has been widely studied in prior research, bringing together expertise in many fields. Switzerland is organized in a federal, decentralized structure of independent governmental entities. Thus, its decentralized structure is a real-world example for implementing an electronic voting system, where trust is distributed among multiple authorities.

This work outlines the design and implementation of a blockchain-based electronic voting system providing cast-as-intended verifiability. The generation of non-interactive zero-knowledge proofs of knowledge enables every voter to verify the encrypted vote, while maintaining the secrecy of the ballot. The Public Bulletin Board (PBB) is a crucial component of every electronic voting system, serving as a publicly verifiable log of communication and ballots - here a blockchain is used as the PBB. Also, the required cryptographic operations are in linear relation to the number of voters, making the outlined system fit for large-scale elections.

CCS CONCEPTS

• **Applied computing** → **Voting / election technologies**; • **Computer systems organization** → **Peer-to-peer architectures**; • **Security and privacy** → *Public key encryption*;

KEYWORDS

Cast-as-Intended Verifiability, Blockchain-based electronic voting

ACM Reference Format:
Christian Killer, Bruno Rodrigues, Raphael Matile, Eder Scheid, Burkhard Stiller. 2020. Design and Implementation of Cast-as-Intended Verifiability, for a Blockchain-based Voting System. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3341105.3373884

1 INTRODUCTION

The electronic transformation of political systems has proven to be a challenging task in recent years [23]. The notion of secure, verifiable, and auditable Remote Electronic Voting (REV) systems has attracted a lot of attention in both research and industry [3]. Retaining privacy whilst achieving verifiability has become a key challenge towards the design and implementation of secure REV systems [19]. Serving as a transparent, immutable, and distributed ledger, Blockchains (BC) offer new benefits for REV [11]. Instead of relying on a single, centralized authority, a system of distributed, equivalent authorities is proposed. Further, BCs serve as a highly replicated, tamper-proof audit trail, which enables the verification of cryptographic proofs, crucial for REV.

Switzerland is formed by a collection of decentralized, independent legal entities (cantons), coordinated under the Swiss Federal Government. Each canton contains multiple municipalities and defines its own constitution and laws. The political system is under the authority of the cantons, *i.e.*, cantonal laws and ordinances regarding political rights are defining elements these processes.

Instead of relying on existing public *permissionless* BCs, the deployment of a public *permissioned* BC is more suitable, since only authorized entities (*e.g.*, Election Authorities (EA)) should be authorized to sign blocks, whilst the general public can verify the BCs data. Also, democratic voting requires central coordination, at least up to the extent of agreeing on the voting question, the voting period, and the decision regarding a voter's eligibility. Thus, permissionless consensus does not provide any desirable properties. Permissioned consensus, on the other hand, enables such things on the protocol layer. Often, the role of a EA can be split up among a set of different electoral parties, such as counties, cantons, or even municipalities. Similarly, a scenario where representatives of political parties cooperate to administer voting is possible. Thus, mirroring the federalistic structure of government entities in a decentralized architecture is a valid approach, which also enables opportunities beyond REV.

A crucial component in a REV system is a secure Public Bulletin Board (PBB). Normally, a PBB serves as a consistent, append-only, publicly available and verifiable log of communications [20]. Thus, a blockchain is perfectly suitable as a PBB. A verifiable system has to be auditable in order to offer verifiability of correct functionality. With a BC-based solution, voters are able to rely on a transparent, immutable, and decentralized ballot box.

While prior work summarized the design, this paper presents the design, implementation and evaluation of a BC-based REV system providing Cast-as-Intended Verifiability [24].

This paper is structured as follows. Section 2 discusses the background and related work. Section 3 details the key assumptions, design, protocol, and implementation. Section 4 includes evaluations, and Section 5 draws conclusions and outlines future work.

SAC '20, March 30-April 3, 2020, Brno, Czech Republic

ACM ISBN 978-1-4503-6866-7/20/03…$15.00
https://doi.org/10.1145/3341105.3373884

2 ENVIRONMENT

This work implements REV in an uncontrolled environment, *i.e.* over the internet. The REV system requires at least two interacting stakeholders in any vote or election: *Election Authorities* (EA) which operate a component to manage, store, and tally votes and Voting Software Client (VSC) executed on the *Voters*' end-clients, such as a computer or mobile device. The following background focuses on the practical security aspects, thus leading towards related work and the actual design.

2.1 Background

In Switzerland, cantons and municipalities are independent bodies [13]. Hence, they can define details of voting methods independently and according to their cantonal constitution [15]. The proposed REV system leverages this federalistic structure and distributing trust among EAs. In order to guarantee a secure voting process, *Sender-anonymous* channels and *untappable communication* channels are required [19]. *Sender-anonymous* channels assure that a sender cannot be identified by the receiver of a message, whereas *untappable communication* channels ensure that neither the sender nor the receiver "can learn anything about the communication, including whether communication occurred or not" [19]. Further, the security of the Voting Software Client (VSC) is crucial for the integrity of the REV system. Two modes of deployment are possible for REV over the internet.

(1) The VSC is executed on secured infrastructure, controlled by the EA.
(2) The VSC is distributed to the Voters, executing it on their devices.

Considering deployment method (2), where the EA provides the VSC binary to the Voter, not only the integrity of the VSC binary needs to be verified, but also the communication channels also need to be secured. Serving the VSC via the internet allows for multiple threat events, such as malicious browser extensions [10]. Possible countermeasures include integrity verification by either signing the code using public key cryptography (which is impractical due to missing standards) or by utilising the W3C Subresource Integrity (SRI)[30] recommendation to verify the integrity of JavaScript code rendered by the browser. Current REV systems use these approaches [10]: First, secure connections to the browser are enforced when loading the VSC code. Second, integrity checks on the server-side are executed. In addition, a dedicated JavaScript application downloads the source code and verifies the code with respect to a previously generated baseline [10].

Further, transaction censorship by malicious nodes should be considered. A malicious node could simply destroy and not forward the vote at all. However, the incentive for such behaviour is low: If the vote is encrypted and the voter's identity is not disclosed to the node receiving the vote (e.g. by using masked identifications). Additionally, broadcasting to multiple nodes mitigates this risk successfully.

2.2 Related Work

Research on secret ballot voting schemes span nearly four decades, continuously refining theoretical properties to evaluate systems and protocols. The notions of privacy evolved from ballot-privacy [8] and receipt-freeness [5] toward coercion-resistance [21] and everlasting privacy [22]. As key building blocks of many REV systems, homomorphic encryption schemes have to be considered, which enable the voter to encrypt votes, while allowing for the tallying of encrypted votes by the authorities, decrypting only the final tally [1]. Additional methods include re-encryption, blind signatures [9], zero-knowledge proof systems, and designated verifier proofs or mixnets [19].

In the context of REV, verifiability states that a voter can trace the effect of his or her vote on the final tally [19]. Ensuring receipt-freeness and coercion-resistance, cryptographic protocol systems are also able to satisfy individual (IV) and Universal Verifiability (UV) [19].

While IV guarantees that a voter can verify that her vote counts correctly [27], UV ensures that anyone can verify that the result is a correct set of votes cast [19]. Additionally, UV guarantees that it is possible to publicly verify that the tally of the ballots is correct. A crucial collection of security properties in REV is End-to-End Verifiability (E2E-V) [4]. E2E-V consists of three core steps [2]: Cast-as-Intended, Recorded-as-Cast and Counted-as-Recorded verifiability.

Thus, achieving CaIV is a first step towards E2E-V in blockchain-based REV. These concepts of privacy and verifiability are continuously challenged and applied to assess electronic voting systems and protocols alike. Prior work focused on designing systems on top of existing permissionless BCs. [26] presents a boardroom voting solution based on an Ethereum Smart Contract as a PBB in which all encrypted votes are stored. [31] applies a ring-signature based approach, using the limited OP-CODE execution environment of the Bitcoin BC. However, the limitations of the execution environments provided by public permissionless BC make it hard to implement and deploy complex cryptographic operations and led to the design and implementation of this work.

3 DESIGN AND IMPLEMENTATION

The following sections describe the relevant assumptions for the proposed REV system. Since this work uses a BC as PBB, the relevant consensus mechanism is detailed as well. Further, the details of the voting protocol enabling Cast-as-Intended Verifiability is outlined, combined with the implementation details.

3.1 Assumptions

Assumptions directly impact any systems architecture, since design decisions must be compliant [7]. Explicitly documenting assumptions prior to the system implementation is key. Thus, the central assumptions are collected in Table 1, taking into account some of the legal requirements of Switzerland.

Considering $A1$, in any election performed in Switzerland, every eligible voter is restricted to cast only a single vote [14]. This contrasts with some theoretical protocols and approaches to coercion-resistant systems, which allow people to cast multiple votes but only count the last submission [3]. To enable $A1$, $A2$ requires every voter to be uniquely identifiable. This does not necessarily imply publicly verifiable links between voters and their votes. Since the number of Swiss cantons and municipalities seldomly change, the

Table 1: Assumptions

ID	Description
*A*1	Every voter is only allowed to cast a single vote, once.
*A*2	Every eligible voter is uniquely identifiable.
*A*3	The voting authority nodes are fixed and known a priori.
*A*4	The connection between the voter's voting device and the voting network is confidential.
*A*5	The submitted votes contain either 1 or 0, thus the votes are binary.

EAs are known in advance (*A*3). Further, *A*4 states that communication channels to the REV system must be confidential. And finally, *A*5 declares the scope of this work, focusing on binary votes.

3.2 Public Bulletin Board (PBB)

A Public Bulletin Board (PBB) is a crucial part of any REV system, because auditable information is published during the execution of the voting protocol. Such a PBB shall provide the following properties[19, 20]:

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

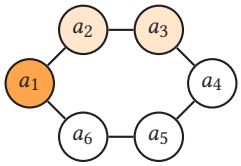
Thus, in the case of Switzerland, mirroring the federalistic governance, a publicly verifiable BC, without one single leader can serve as an appropriate PBB.

3.3 Consensus Algorithm

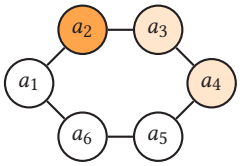
Selecting an appropriate byzantine fault-tolerant (BFT) consensus algorithm in a permissioned BC is crucial for the security and performance properties of the proposed REV system. Ethereum's Proof-of-Authority (PoA) consensus algorithm *Clique* [29] provides a suitable and efficient approach for a public permissioned BC. This work applies a simplified version of the original Clique [29] algorithm, since it does not include mechanisms to add or remove authorized signer nodes, as according to Assumption 3, this functionality is not desirable for a REV system where all EAs are known beforehand.

3.3.1 Leader and Co-Leader. In the Clique consensus algorithm, epochs are fixed time periods. During epochs, a set of nodes is authorised to sign and broadcast blocks: One leader and a number of co-leaders. Nodes recognize their role by calculating whether their respective index in the list of authorities is considered a leader or co-leader for the current epoch. Figure 1 shows an example where the block period is set to two, so in that instance, two co-leaders are always defined during each epoch.

Upon receiving a new transaction, t_n, nodes validate parameters of t_n for correctness, while also verifying that t_n is not yet persisted in the local storage to avoid processing duplicate transactions. If the node is leader or co-leader for the current epoch, the node appends t_n to the local storage $\mathcal{T}$, which contains a pool of all transactions waiting to be summarized to a block. Further, the received transaction is broadcast to all to propagate the transaction



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



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

Figure 1: Clique's leader election with signer limit set to two

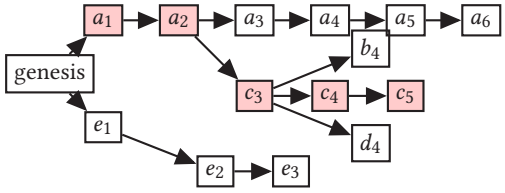


Figure 2: A chain of blocks with its heaviest chain marked in red

to other peers and EAs, which also assures that race conditions between diverging transaction pool states among leaders and co-leaders are avoided. Similarly, upon receiving a new block, the block is verified for its validity, *i.e.* whether the block is correctly signed and whether the references to the transactions are correct. Eventually, the new block is appended to the canonical chain.

Blocks are only signed if the node is currently a leader or co-leader. Then, the node waits until the period of the current epoch has ended and immediately builds the block with all known transactions. To reduce the number of forks occurring when both the leader and its co-leaders announce a block at the same time, co-leaders are urged to delay their broadcast by a small amount of time, allowing the other nodes to receive the block from the leader first.

As multiple nodes are allowed to propose blocks per epoch, BC forks can occur [12]. In Ethereum, the Greedy Heaviest-Observed Sub-Tree (GHOST) protocol [28] is used to resolve such conflicts. In short, it chooses the heaviest subtree of each block for building the main chain for a specific notion of weight. By assigning weights to each block, such that the block of the leader weighs more than the ones of its co-leaders, the main chain can be constructed. Further, so-called uncle blocks (all other children of the grandparent of the block in consideration), and blocks which were already referenced as parents multiple times, will also be preferred. For instance, considering the BC depicted in Figure 2 and suppose the GHOST protocol considers the number of direct uncles and children for the weight assignment. Starting from the blocks with the lowest height, node c_4 has a weight of 2, as it has one child and one uncle (a_3). Node c_3 has a weight of 3, as it has three children but no uncle. The weight for node a_3 equals the amount of its children, i.e. 1. Hence, the decision on the branch at a_2 is made in favour for c_3 as its weight, 3, is heavier than the one from a_3, which is 1.

To further give preference to blocks which have been signed by a leader, Clique specifies a doubled weight for blocks being signed by a leader. In particular, Ethereum's Clique implementation is

calculating up to seven levels[1] ahead for computing the number of uncles of a block.

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

A block represents the body in which a set of transactions are contained. A block contains a timestamp, showing when it was created, the transactions as payload, and the parent hash, identifying a block as its parent. Based on that, the BC can be built. The genesis block is a special block instantiation and specifies all system-relevant parameters for the BC. As such, it is used as the root of the BC and the hash of its contents forms the first reference to which all direct child-blocks refer. Any change applied to the configuration of the genesis block or to the content of all other blocks will lead to a different hash and, therefore, a different chain of blocks.

3.4 Voting Protocol Design

An overview of the four-step protocol [17] is provided in Figure 3.

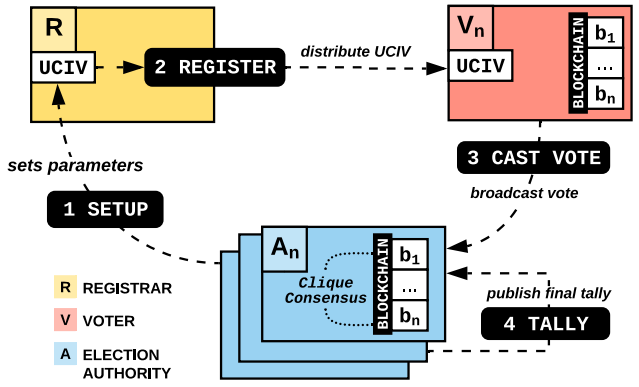


Figure 3: System overview

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

The *Setup* protocol [17] generates a public-private key-pair (pk_e, sk_e) for the additive ElGamal cryptosystem [16]. (a) The space of the secret Universal Cast-as-Intended Verification (UCIV) information (SVI), (b) the corresponding space of voting-option dependent secret UCIV information $\widetilde{SVI}$, and (c) the space of public UCIV information PVI are generated. The function $\sigma_v : SVI \rightarrow \widetilde{SVI}$ mapping the secret UCIV information to a voting option-dependent secret UCIV information is specified as well.

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

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

The *Register* protocol [17] generates the private and public UCIV information for a particular voter, based on the voter id vid, the set of voting options V, and the public election key pk_e. The *Register* protocol is executed by a registrar independent of the voting authorities, assigning the voter id vid to all voters. In a scenario where election authorities could link from the vid to a voter's real-world identity, authorities could decrypt votes and break ballot secrecy since the authorities possess the election private key sk_e. Depending on the trust assumptions and requirements, the registrar can be a state-owned entity, but should be separated from the election authorities to minimize the risk of collusion.

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

Step 3 starts as soon as the election is opened (*cf.* Figure 4, third vertical dashed line). An indicator is published on the PBB, specifying that ballots are accepted now. To generate a valid ballot, (a) the voter id vid, (b) the selected voting option v, and (c) the election public key pk_e are required. In addition, (d) the output of the evaluation of the voting option-dependent function on the secret UCIV information $\sigma_v(uciv_s)$, and (e) the corresponding public UCIV information $uciv_p$ is provided as protocol input. With the help of these arguments, the homomorphic additive ElGamal ciphertext C^{vid} is generated. Correspondingly, a universal Cast-as-Intended verification proof V^{vid} is generated [17]. A membership proof guarantees that the encrypted vote either represents a zero or one, ensuring that only valid binary votes are submitted to the PBB. This proof is added as M^{vid} along the cast-as-intended proof and the encrypted vote to the ballot. Finally, the ballot is cast to the PBB.

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

Before election authorities determine the final tally, the election must be closed. In order to calculate the final tally, the set of all triples $(C^{vid}, M^{vid}, V^{vid})^*$ on the PBB and the election's private key sk_e are required. With homomorphic addition of all valid ciphertexts, the total amount of supporting ballots is obtained as T_s. By subtracting the total supporting ballots from the total number of ballots received, the opposing vote count is determined as T_o. Invalid ballots, which are excluded from the final tally (*e.g.*, because of invalid proofs) are counted as T_i. Thus, the triple (T_s, T_o, T_i) represents the final tally T.

The stakeholders' step-wise involvement during the voting protocol is detailed in Figure 4. These stakeholders include: The Election Authorities (A), such as districts or cantons, the Registrar (R), authenticating eligible voters according to the electoral register, the Voter (V), who desires to submit his or her vote, the voting device, which is operated by the voter in order to cast a ballot, and the PBB.

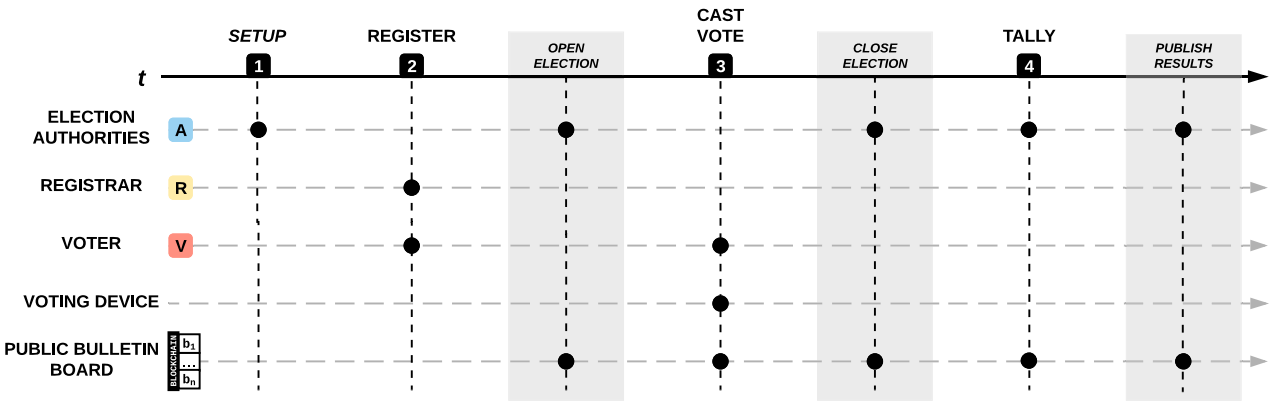


Figure 4: Involvement of stakeholders in voting protocol steps

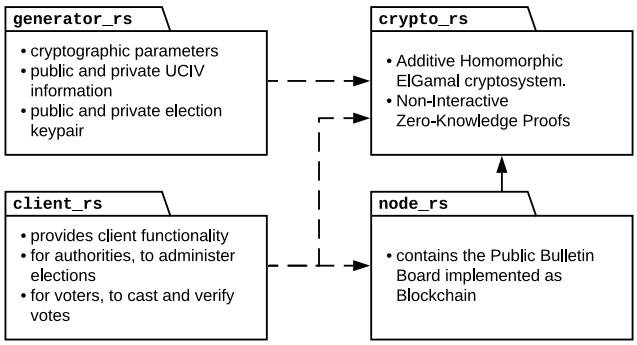


Figure 5: System components

3.5 Implementation

The system is structured into four modules implemented in Rust [25] (*cf.* Figure 5). The `crypto_rs` package provides the arithmetic primitives for the additive homomorphic ElGamal cryptosystem and the non-interactive zero-knowledge proofs (NIZKP). The `generator_rs` package generates the cryptographic parameters required, such as the private and public UCIV information and the election public-private key-pair. The `node_rs` package provides the PBB BC implementation, whereas the `client_rs` package provides the functionality to govern and participate in the election.

In detail, a data structure based on `ModInt` provides modular arithmetic operations. The additive variant of the homomorphic ElGamal ciphertext is defined by G, H, and the random number r used while encryption is performed. In this regard, the encryption, decryption, and homomorphic additions are implemented as follows: By defining the message space of all valid plain-text votes to be in the cyclic subgroup G of order q of $(\mathbb{Z}_p)^*$, with q being co-prime to p and g being the generator of G. The aforementioned processes are mapped to the following steps:

(1) Generate a *private key* by selecting a random number x from the uniformly distributed set $\{1, ..., q-1\}$ and keep x secret.
(2) Generate the corresponding *public key* by calculating $h = g^x$. Make the set (G, q, g, h) public.
(3) Encrypt a message $m \in \mathbb{Z}_p$ using $r \in_{\text{uniform}} \mathbb{Z}_p$ with the public key h by calculating the shared secret $s = h^r = (g^x)^r = g^{xr}$. Then, the resulting ciphertext is defined as $E(G, H) = (g^r, g^m \cdot s)$.
(4) Decrypt a ciphertext $E(G, H)$, by recalculating the secret $s = G^x = (g^r)^x = g^{rx}$ and $g^m = H \cdot (s^{-1}) = g^m \cdot h^r \cdot (g^{xr})^{-1} = g^m \cdot g^{xr} \cdot g^{-xr}$ with s^{-1} being the modular multiplicative inverse of s. Then, solve the discrete logarithm to obtain m.

After obtaining two ciphertexts $E(m_1)$ and $E(m_2)$, the homomorphic addition can be performed as follows:

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

In order to transform the ElGamal range proof to its non-interactive form, the Fiat-Shamir heuristic [18] is used. Similar to the range proof, [17] defines the Cast-as-Intended verification proof in its non-interactive form. The `generator_rs` binary is able to generate all required cryptographic material, such as the private and public UCIV information and the election key-pair. However, creating a new ElGamal key-pair is not yet cryptographically safe: As of today, safe prime generators are unavailable for the `BigInt` abstraction used. The prime modulus p, its co-prime q, and the private key x are currently hard-coded. The `generator_rs` represents a best-effort solution to enable the end-to-end voting process. In addition to generating an election key-pair, the `generator_rs` generates a set of public and secret UCIV information $(uciv_s, uciv_p)^*$. In the current implementation, the voting option dependent function σ_v is already applied to the set of secret UCIV information $uciv_s$. Since the computation of a logarithm of a safe prime is considered computationally expensive in the ElGamal cryptosystem, σ_v represents the exponentiation function $F(x) = g^x$ as proposed in [17].

`node_rs` contains the implementation of the PBB as a BC. Thus, `crypto_rs` is required as a dependency, providing necessary data structures to build transactions and blocks. Figure 6 shows its components: **I** BC nodes communicate using the `Node RPC interface`. **II** Votes are submitted to a dedicated `Client RPC interface`. Two

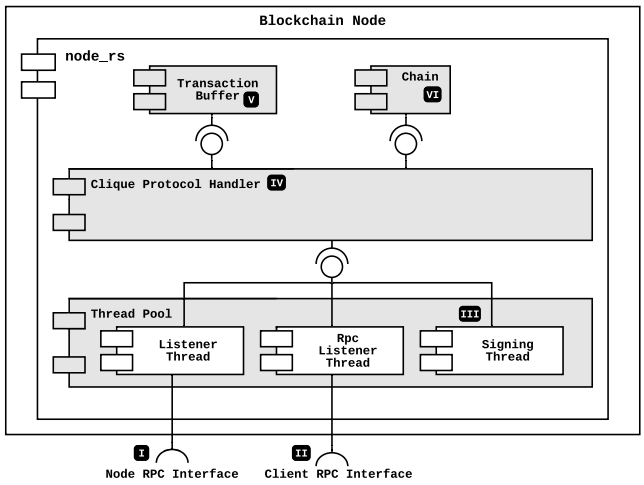


Figure 6: The blockchain node's architecture.

threads listen to incoming connections, while a thread pool III executes another thread, signing blocks, if the node is a leader or co-leader. The `Clique Protocol Handler` IV operates on the Proof-of-Authority (PoA) level, handling incoming messages and the corresponding responses, creating transactions and blocks, and determining whether the node is a leader or co-leader for the current epoch. In addition, it holds a transaction buffer V and its own instance of the actual BC data VI. Since the three threads share the same instance of the protocol handler, a mutex avoids race conditions. Thus, all threads base decisions on the same BC instance, also illustrated by the single interface to the `Clique Protocol Handler` in Figure 6.

The `configuration` module contains utilities to read required configuration parameters, which are used as a genesis block for the BC. Besides a version flag, the block period for the Clique protocol [29], the number of blocks each node in the network is allowed to sign consecutively, the election public key pk_e. The public UCIV information $uciv_p$ is required to verify the proof of any transaction. During the runtime instantiation of the Clique protocol, the genesis block hash is used to identify whether a node in the network is based on the same configuration. If a configuration value is different, the hash will also change, and thus, nodes will never agree on the same canonical chain. In conclusion, nodes with a different genesis block hash are excluded from communications.

Instead of using a tree-based data structure to build up the chain, an adjacent matrix is created, containing all block identifiers on the y-axis and the corresponding children identifiers of the block on the x-axis. This ensures an $O(N)$ cost when looking up a particular key. The BC data is stored on the heap.

The `p2p` module includes the foundation of `node_rs`: It defines TCP (Transmission Control Protocol) streams as connection channels between nodes, a thread pool in which multiple tasks can be handled concurrently, and a codec transforming incoming and outgoing messages into their appropriate formats. As shown in Figure 6, each `node_rs` instance owns three threads in which the main operations are applied to a shared instance of the BC: *(a)* the `Listener Thread` responding to incoming node connections, *(b)* the `RPC Listener Thread` answering requests from client applications, and *(b)* a thread signing blocks.

Nodes use JSON (JavaScript Object Notation) to exchange information. Thus, all messages are encoded to JSON by using an appropriate codec before they are serialized into a byte sequence and sent over the underlying TCP streams. The serialisation format can be adapted without influence on the rest of the implementation of `node_rs`, since it is defined in a separate package.

Communication between nodes strictly follows the protocol defined in the `Clique Protocol Handler`. As clients who are submitting votes expect different responses than other nodes in the BC, dedicated interfaces handle interactions, either the `Listener Thread` and `Rpc Listener Thread`, respectively. Transactions received by a node in the BC network are always broadcast to other nodes, regardless of their actual payload. Therefore, not only will epoch leaders be notified upon receival of a new transaction, their co-leaders are also notified. Besides the block and transaction broadcasts, the `Clique Protocol Handler` can send a full copy of its own chain to other nodes. During start up, each node will broadcast a `ChainRequest` to its other peers, which are defined in the genesis configuration. On request, a node returns a full copy of its BC data to the requester, which replaces its own BC data, *iff* the genesis hash is equal to its own, and the depth of the canonical chain is longer than its own. The client side functionality to administer elections, to submit votes, and to obtain a final tally is accessible through the `client-rs` application.

3.6 Vote Administration

Election authorities open and close the election by sending an `OpenVote` and `CloseVote` message to the PBB respectively. Incoming vote transactions are only included in the final tally, if the election is open. To submit a vote, voters need to be in possession of the election public key pk_e and their associated private and public UCIV information pair $(uciv_s, uciv_p)^{vid}$, which they received from the Registrar. Then, they can cast either `yes` or `no`, answering the voting question. The voting choice is transformed to binary (1 and 0) and encrypted on the voter's client device. The range proof and the Cast-as-Intended verification proof are generated. Before submitting the vote to the BC, both proofs are validated on the voter's client. Finally, if validation was successful, the vote will be broadcast to the nodes of the peer-to-peer BC network. Once the election is closed by the authorities, the final tally can be determined. By sending a `RequestTally` to a BC node, a traversal from the root to the end of its current canonical chain is initiated. Each block's vote transactions T_V are homomorphically summed up. Before that, each vote transaction is validated with their associated proofs. If the proof verification fails, the corresponding transaction is counted toward the invalid vote count. Once the entire canonical chain is traversed, a response containing the amount of successful, invalid, and total votes is returned. If no `CloseVote` transaction is observed during the traversal of the canonical chain, zero values are returned for the above parameters.

4 EVALUATIONS

The *Cast-as-Intended* verifiability allows voters to verify that their encrypted vote contains the selection they made. By generating

and verifying non-interactive zero-knowledge proofs of knowledge before submission to the PBB, voters are ensured that the vote is appropriately encrypted. Considering a scenario, where an adversary distributes a malicious binary to the voter, modifying a binary to always show a successful Cast-as-Intended verification proof regardless of the voter's choice, a voter can always detect such case. As of today, it is considered the standard that software vendors provide signed binaries and corresponding checksums, which can be used to verify the integrity of the executed binary.

4.1 Performance and Results

For real-world elections, the performance to generate the cryptographic parameters is essential. Optimally, the runtime is linear in the number of voters, *i.e.*, $O(n)$ for n voters. The runtime complexity is linear to the number of voters (*cf.* Figure 7a. The time required to generate the UCIV information behaves linearly in the number of voting options. Figure 7b shows the linear behavior of the runtime when generating the UCIV information for multiple amounts of voters. Based on these numbers, a linear approximation to generate UCIV information for the average Swiss electorate of 5, 357, 836 eligible voters in 2017[6] leads to an approximate of 743 s (∼ 12 min) for two voting options, 1,069 s (∼ 18 min) for three voting options, and 1,464 s (∼ 24 min) for four voting options.

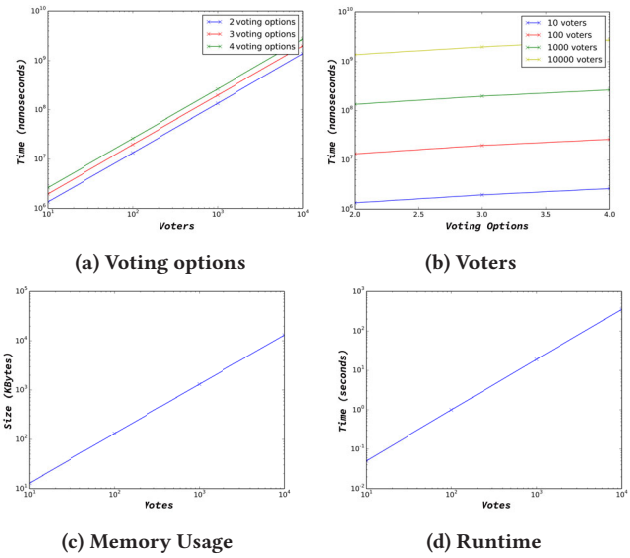
Figure 7c indicates the storage requirements to store 10, 100, 1,000, and 10,000 vote transactions T_V with their corresponding proofs. The storage is also linear to the number of transactions. Finally, Figure 7d shows the time required to handle these votes by the Clique protocol. Since the `Clique Protocol Handler` verifies, whether a transaction is already known, the set of known transactions is queried, *i.e.*, the performance is also linear to the number of transactions.

Thus, obtaining the final tally is achievable in a timely manner: considering a BC with 10, 90, 900, and 9,000 transactions in consecutive blocks: the result of an election can be retrieved in less than one second.

4.2 Discussion and Limitations

Besides *Cast-as-Intended* verifiability, *Recorded-as-Cast verifiability* can be ensured as well: If a voter obtains the identifier of the transaction submitted to the BC, the BC can be queried after the vote's submission. Thus, the voter is able to verify the associated proofs for correctness. Nevertheless, since transactions are not signed with a voter-dependent value, the integrity verification of the vote becomes poor. However, the transaction's integrity can be verified by computing the hash of the transaction before its submission to the BC. *Individual verifiability* is provided since the voter is able to query the BC and evaluate it for correctness using the associated proofs. *Counted-as-Recorded verifiability* is not yet provided by the design. However, the voting protocol can be extended with the computation of a non-interactive zero-knowledge proof of knowledge, proving the correct tabulation, and publishing it to BC as well. Thus, universal verifiability can be provided by publishing a tabulation proof of the final tally to the BC.

Although votes are locally encrypted on the voters' end-user devices, election authorities are still able to decrypt individual votes, if there's a function to query the BC for individual transactions.



(a) Voting options (b) Voters

(c) Memory Usage (d) Runtime

Figure 7: Runtime for generating public and private UCIV information $(uciv_s, uciv_p)^*$ as well as storage and runtime evaluations

As this functionality is necessary to provide *Recorded-as-Cast* verifiability to voters, it makes sense for election authorities to have access as well, especially as they provide the infrastructure running the BC. The risk of vote decryption can be mitigated by using multiparty computation [17]. Moreover, the confidentiality of the vote is not broken as the voter id *vid* is only linked to an identity by the registrar. As long as the registrar behaves honestly and independently from the election authorities, and collusion can be avoided, ballot secrecy is assured because the link between voter and vote is only checked for by the registrar.

Once the public and private UCIV information $(uciv_s, uciv_p)^{vid}$ set and the election public key pk_e are obtained, only one step is necessary to submit a valid vote, which is comparable to casting a physical ballot via postal mail. With the additional introduction of electronic identity solutions, the initial complexity of the registration step for a voter can be reduced because the process can be digitized and simplified. It is possible that the UCIV information $(uciv_s, uciv_p)^{vid}$ is obtained over electronic channels (*e.g.*, encrypted e-mail) from an independent set of registrars, rendering printouts for paper-based voting signature cards and paper ballots fully obsolete.

The Clique Consensus Algorithm can tolerate up to $\frac{N}{2} - 1$ malicious participants [12]. Thus, the voting system can reach consensus with dishonest election authorities if N is large enough. Since voters can query the PBB for their vote submitted, a malicious authority censoring arbitrary transactions by abstaining from broadcasting them to other peers can be identified (by querying whether the transaction was included in the BC). In such a case, voters can resubmit their vote to another authority and query the PBB for their vote, being assured that the vote was cast. A non-negligible limitation of this current implementation is caused by the homomorphic addition of ciphertexts on the BC: The randomness used in each

ciphertext must be provided for correct homomorphic tabulation. Since this calculation is executed on a node running an instance of the PBB, the randomness is submitted along with the encrypted vote and the corresponding proofs to the BC. However, by making this information public, an adversary obtaining confidential information can cause a significant shrinkage of the solution space it has to search through to decrypt a ciphertext.

5 CONCLUSIONS AND FUTURE WORK

This work on Cast-as-Intended verifiability details the implementation and evaluation of a suitable and operational approach to provide BC-based electronic voting. With the storage of NIZKPs on the BC, the Cast-as-Intended verifiability property can be fulfilled, while at the same time, privacy is maintained. In addition, tabulation of the final tally is performed directly on the BC, which distributes trust among multiple authorities.

Concluding, opposing properties of verifiability and privacy also revealed challenges in providing Recorded-as-Cast and Counted-as-Recorded verifiability in a distributed setup, *i.e.*, ballot privacy can potentially be reduced by homomorphic tabulation directly on the BC. Thus, with respect to the Swiss scenario, further extensions to the current proof-of-concept implementation are necessary in order to fulfill the end-to-end verifiability.

Six additional aspects require attention for future work: *(a)* Performing the homomorphic addition on the BC leaks information about the randomness of the ciphertexts, thus reducing the solution space an adversary needs to consider when trying to decrypt a vote. *(b)* The current prototype omits authentication of incoming RPC messages. Incoming messages need to be signed with an asymmetric key-pair by each election authority. Thus, illegitimate requests can be declined. *(c)* Consider additional properties to increase verifiability and privacy, *e.g.*, providing everlasting privacy and coercion-resistance [22] *(d)* Additionally, multiparty computation could distribute a composite private key across multiple authorities, requiring a threshold of k out of n authorities in order to decrypt any value. *(e)* Further, the voter ID (*vid*) should be replaced by blinded tokens to reduce the risk of collusion between the registrar and the EA. *(f)* Lastly, to increase voter privacy, Onion Routing could be included, where individual votes are first encrypted with the public keys of multiple authorities, relaying them before they are included in the BC.

ACKNOWLEDGMENTS

This paper was supported partially by (a) the University of Zurich UZH, Switzerland and (b) the EU H2020 Program under Grant Agreement No. 830927 (Concordia). The authors would like to thank Sarah Jamie Lewis for the feedback on the final version.

REFERENCES

[1] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. 2018. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. *ACM Computing Surveys (CSUR)* Vol. 51, No. 4 (July 2018), pp. 79:1–79:35.

[2] Syed Taha Ali and Judy Murray. 2016. An Overview of End-to-End Verifiable Voting Systems. arXiv:1605.08554 (May 2016). http://arxiv.org/abs/1605.08554

[3] Josh Benaloh, Matthew Bernhard, J. Alex Halderman, Ronald L. Rivest, Peter Y. A. Ryan, Philip B. Stark, Vanessa Teague, Poorvi L. Vora, and Dan S. Wallach. 2017. Public Evidence from Secret Ballots. arXiv:1707.08619 (August 2017). http://arxiv.org/abs/1707.08619

[4] Josh Benaloh, Ronald L. Rivest, Peter Y. A. Ryan, Philip B. Stark, Vanessa Teague, and Poorvi L. Vora. 2015. End-to-end Verifiability. arXiv:1504.03778 (2015). http://arxiv.org/abs/1504.03778

[5] Josh Benaloh and Dwight Tuinstra. 1994. Receipt-free Secret-ballot Elections. In *26th Annual ACM Symposium on Theory of Computing, (STOC 1994)*. pp. 544–553.

[6] Bundesamt für Statistik. 2019. Stimmbeteiligung. http://bcbev.ch/turnout last visit December 1st, 2019.

[7] Janet E. Burge and David C. Brown. 2008. Software Engineering Using RATionale. *Journal of Systems and Software* Vol. 81, No. 3 (March 2008), pp. 395–413.

[8] David Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* Vol. 24, No. 2 (February 1981), pp. 84–90.

[9] David Chaum. 1983. Blind Signatures for Untraceable Payments. In *Advances in Cryptology*. Springer US, Boston, MA. U.S.A., pp. 199–203.

[10] Jordi Cucurull, Sandra Guasch, and David Galindo. 2016. Transitioning to a Javascript Voting Client for Remote Online Voting. In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016)*. SCITEPRESS - Science and Technology Publications, Lda, Portugal, pp. 121–132.

[11] Jordi Cucurull, Adrià Rodríguez-Pérez, Tamara Finogina, and Jordi Puiggali. 2018. Blockchain-Based Internet Voting: Systems' Compliance with International Standards. In *Business Information Systems Workshops - (BIS 2018) International Workshops, Berlin, Germany*. pp. 300–312.

[12] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. 2018. PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain. *Italian Conference on Cyber Security, (CEUR 2018) Workshop* (February 2018), pp. 1–11.

[13] Die Schweizerische Bundeskanzlei. vom 13. Mai 2015. Umsetzung von Artikel 50 der Bundesverfassung. http://bcbev.ch/fg. last visit December 1st, 2019.

[14] Die Schweizerische Bundeskanzlei. vom 24. Mai 1978 (Stand am 15. Januar 2014). Verordnung über die politischen Rechte (VPR). http://bcbev.ch/vpr. last visit December 1st, 2019.

[15] Die Schweizerische Bundeskanzlei (1). vom 17. Dezember 1976 (Stand am 1. November 2015). Bundesgesetz über die politischen Rechte (BPR). http://bcbev.ch/bpr. last visit December 1st, 2019.

[16] Taher Elgamal. 1985. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory* Vol. 31, No. 4 (1985), 469–472.

[17] Alex Escala, Sandra Guasch, Javier Herranz, and Paz Morillo. 2016. Universal Cast-as-Intended Verifiability. *Lecture Notes in Computer Science* 9604 LNCS (August 2016), pp– 233–250.

[18] Amos Fiat and Adi Shamir. 1987. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology (CRYPTO '86)*, Andrew M. Odlyzko (Ed.). Springer, Berlin Heidelberg, pp. 186–194.

[19] Hugo Jonker, Sjouke Mauw, and Jun Pang. 2013. Privacy and Verifiability in Voting Systems. *Computer Science Review* Vol. 10 (November 2013), pp. 1 – 30.

[20] Hugo Jonker and Jun Pang. 2011. Bulletin Boards in Voting Systems: Modelling and Measuring Privacy. *Proceedings of the 2011 6th International Conference on Availability, Reliability and Security, (ARES 2011)* (2011), pp. 294–300.

[21] Ari Juels, Dario Catalano, and Markus Jakobsson. 2010. *Coercion-Resistant Electronic Elections*. Springer Berlin Heidelberg, pp. 37–63.

[22] Philipp Locher, Rolf Haenni, and Reto E. Koenig. 2016. Coercion-Resistant Internet Voting with Everlasting Privacy. In *Financial Cryptography and Data Security*. Springer, Berlin Heidelberg, pp. 161–175.

[23] Harald Mahrer and Robert Krimmer. 2005. Towards the Enhancement of e-Democracy: Identifying the Notion of the 'Middleman Paradox'. *Information Systems Journal* Vol. 15, No. 1 (January 2005), pp. 27–42.

[24] Raphael Matile, Bruno Rodrigues, Eder Scheid, and Burkhard Stiller. 2019. CaIV: Cast-as-Intended Verifiability in Blockchain-based Voting. *1st IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2019)* (May 2019). Seoul, South Korea.

[25] Nicholas D. Matsakis and Felix S. Klock, II. 2014. The Rust Language. In *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology (HILT '14)*. ACM, New York, NY. U.S.A., pp. 103–104.

[26] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. 2017. A Smart Contract for Boardroom Voting with Maximum Voter Privacy. In *Financial Cryptography and Data Security*, Aggelos Kiayias (Ed.). Springer, Cham, pp. 357–375.

[27] Kazue Sako and Joe Kilian. 1995. Receipt-Free Mix-Type Voting Scheme. In *Advances in Cryptology, (EUROCRYPT 1995)*. Springer, Berlin Heidelberg, pp. 393–403.

[28] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure High-Rate Transaction Processing in Bitcoin. In *Financial Cryptography and Data Security*, Rainer Böhme and Tatsuaki Okamoto (Eds.). Springer, Berlin Heidelberg, pp. 507–527.

[29] Péter Szilágyi. 2017. Clique PoA protocol & Rinkeby PoA testnet. http://bcbev.ch/eip225. last visit December 1st, 2019.

[30] W3C. 2016. Subresource Integrity, W3C Recommendation. http://bcbev.ch/sri. last visit December 1st, 2019.

[31] Yifan Wu. 2017. An E-Voting System based on Blockchain and Ring Signature. Master's Thesis, University of Birmingham.