Trustless, Censorship-Resilient and Scalable Votings in the Permission-based Blockchain Model[*]

Sebastian Gajek and Marco Lewandowsky[†]

IT Security and Cryptography Group
Flensburg University of Applied Sciences, Germany
`https://www.itsc.inf.hs-flensburg.de`

Abstract

Voting systems are the tool of choice when it comes to settle an agreement of different opinions. We propose a solution for a trustless, censorship-resilient and scalable electronic voting platform. By leveraging the blockchain together with the functional encryption paradigm, we fully decentralize the system and reduce the risks that a voting provider, like a corrupt government, does censor or manipulate the outcome.

Keywords: DAO, voting, functional encryption, blockchain, Hyperledger

1 Introduction

In many countries, the de-facto mechanism to realize democratic choices are votings. Recently, electronic voting systems gained much attention as they have paved the ground for reinventing the financing, controlling and management of organizations. Instead of a central authority, in *Decentralized Autonomous Organizations (DAOs)* [38, 11], people democratically determine the next decision taken by the organization [17]. This resembles the key principle of blockchain technologies, namely to agree on the next state in a decentralized manner. Considering the enormous market caps, the rapid adaption and the impact a voter will have within the organization, building upon a secure, trusted and reliable voting platform is pivotal for a prolonged proliferation of DAOs. It is well-known that voting systems are subjected to attacks which threaten democratic decision-making. Attempts to manipulate elections have occurred regularly within the last years [14]. This includes vote-buying, ballot-stuffing, destruction or invalidation of ballots, mis-recording of votes, aggravating the

[*] This work was supported by the European Commission through the EC H2020 FENTEC under grant agreement No 780108.

[†] Contact author: `marco.lewandowsky@hs-flensburg.de`

Properties	[26]	[18]	[27]	[39]	**This work**
Fairness	✗	✓	✗	✓	✓
Eligibility	✓	✓	✓	✓	✓
Privacy	✓	✓	✓	✓	✓
Individual Verifability	✓	✓	✓	✓	✓
Universal Verifability	✓	✓	✓	✓	✓
Trustlessness	✗	✗	✓	✗	✓
Scalability	✗	✗	✗	✗	✓
Vote Type	any	any	yes-no	any	any

Table 1: Comparison of different e-voting protocols in the blockchain

voting access or tampering with the electronic voting machines [28, 13, 4, 15, 10]. The commonality of all these threats is to affect the outcome of the voting. For important decisions (e.g. presidential or shareholder elections) one tries to reduce the threats by recruiting trusted helpers and the engagement of neutral observers. These entities are appointed by a central authority, like the government or a corporate, and are crucial to the election process. Centralized trust is fragile. Even if their implementation is cost-efficient, the history has shown that central authorities can misuse their responsibility and power to influence the outcome of an election to their favour. Given the scale and frequency at which DAOs ask for democratic decisions, human aided conduction and auditing of the votings is nearly impossible.

1.1 Previous Work

Electronic voting systems and their security properties [16, 32, 34, 31] have been actively studied in the research community, since their introduction in the celebrated work of Chaum [7, 35, 3, 20]. Due to their byzantine fault-tolerance properties [5] blockchain technologies turn out to be a promising tool for electronic voting systems [30, 19].

Liu et al. propose a voting protocol within the permissioned and permissionless blockchain model [26]. Their protocol runs between a voter, central organizer and inspector. A voter casts a vote by encrypting the vote with the organizer's public-key before the inspector signs it. The system thus assumes to trust both parties not to violate ballot and voter privacy. An implementation based on the blockchain framework Ethereum is given in [2]. With regard to today's gas prices and Ethereum's throughput, the system is unsuitable for frequent or large-scale elections.

Hardwick et. al propose a voting scheme in the permission-based blockchain model, satisfying the basic notions fairness, eligibility, privacy and verifiability (see Section 5 for an explanation) [18]. Their protocol uses the blockchain as a transparent ballot box. The system relies on a central certificate authority to authenticate voters and give permission to access the network. Hence, an authority, when byzantine, breaks the link between voter identity and casted vote and therefore violates the ballot privacy. Their implementation within a private Ethereum chain requires a non-negligible amount of gas per vote, which makes the approach less appealing for frequent votings like in DOAs.

McCoy et al. propose a variant of the Open Vote Network (OVN) in a permissionless blockchain model [27]. The OVN is a self-tallying protocol avoiding a central counting authority. A self-tallying protocol converts tallying into an open procedure, that allows any voter or a third-party observer to perform the tally computation once all ballots are casted. This removes the role of a tallying authority in an election as anyone can compute the tally without assistance. Unfortunately, self-tallying protocols have a fairness drawback as the last voter can compute the tally before anyone else which results in both adaptive and abortive issues. Moreover, the protocol is limited to boardroom votes, where board members take a yes-no decision. Their implementation in Ethereum shows suitable gas costs for 40 voters, but does not scale with larger numbers.

Yu et al. propose a platform-independent approach [39]. To achieve the comprehensive goal, the authors employ Paillier encryption to enable ballots to be counted without leaking candidature information in the ballots. To leverage the homomorphic property for summation of votes, an administrator decrypts the plaintext sum. If byzantine, the administrator can use the same decryption key to decrypt each encrypted ballot and break the ballot privacy. A proof-of-knowledge is employed to convince the voting system that the ballot, casted by a voter, is valid without revealing its content. Linkable ring signatures are used to ensure that the ballot is from one of the valid voters, while no one can trace the owner of the ballot. To this end, a voter needs to download the public keys of all other voters, which entails a space allocation linear in the number of voters. Their reference implementation in Hyperledger Fabric allows to handle millions of voters, provided that they are grouped in sufficient batches.

1.2 Our Contribution

We propose a solution for a trustless voting system. By trustless, we mean a system in which no byzantine party, including the voting organizer, manipulates the outcome of the election. A bit more precise, malicious organizers are prevented from opening ballots before the official tallying. We leverage techniques from functional encryption [36, 23] and implement decentralized *off-chain opening oracles*. In our system, voters encrypt their votes and store them in the blockchain. This technique already leads to censorship-resilience of the casted votes, as the blockchain guarantees the immutability of the storage. Off-chain oracles like for example time-triggered servers open the encrypted votes by writing their decryption keys into the blockchain. Only if a sufficient subset of keys, matching a pre-defined quorum policy, has been stored, the blockchain or any other auditing entity is capable of opening and publicly tallying the ballots. This way we fully decentralize the opening phase and lift the byzantine fault-tolerance properties of blockchains to the off-chain perimeter.

In a permission-based blockchain model each voter has a unique identity upon which the blockchain network verifies the user's eligibility to access the network. The risk is that byzantine nodes trace the opened ballot back to the identity of the voter, thus breaking the privacy property of the voting system. One might be tempted to leverage anonymous credentials to prove eligibility in zero-knowledge [6]. While the approach addresses the privacy issue, it opens a new problem. The anonymity allows voters to run a double-voting attack and inject arbitrarily-many votes in the election process. To resolve the eligbility-privacy paradox in the permission-based blockchain model we introduce the notion *off-chain anonymizer oracles* alongside present authentication mechanisms. The anonymizer oracles cooperatively unlink the encrypted vote from the user's identity and enforce the eligibility to cast a single vote. To instantiate the oracles, we leverage techniques from threshold blind signatures [22, 8, 25] where the threshold parameter allows us to scale the byzantine fault-tolerance of the voting system. Only if the eligible voter receives sufficient signatures from the anonymizer oracles, she can unblind and reconstruct an anonymous voting credential that is necessary to cast her encrypted vote.

Because of the additional redundancy of the network, an interesting research question is to investigate the scalability of the voting system. We implement the voting system prototypically in the Hyperledger Fabric framework [1] and evaluate the performance with different network configurations. Our implementation shows that the system is capable of handling large-scale votings with millions of voters like presidential elections or high-frequency votings as they appear in DOAs.

2 Preliminaries

2.1 Notations

$\mathbb{N}$ is the set of natural numbers. A finite set S of elements from 1 to $n \in \mathbb{N}$ is denoted as $S := \{1, \ldots, n\}$. The output b of an algorithm $\mathcal{A}$ with input a is denoted as $b \leftarrow \mathcal{A}(a)$. A vector of length n is written as $\boldsymbol{x} = \{x_1, \ldots, x_n\}$. Further we denote the dot product of two vectors $\boldsymbol{x}, \boldsymbol{y}$ as $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$.

2.1.1 Authenticated message transmission

An authorized identity $\mathcal{I}_A := \{sk_A, pk_A, cert_A\}$ of a user A is a three tuple consisting of a secret key sk_A, a public key pk_A and a certificate $cert_A$, created by an authority over some attributes $attr_A$ and the public key pk_A of A. We write $A \quad \mathcal{I}_A \xrightarrow{msg} \quad B$ to denote the protocol, where a message msg signed with sk_A of the identity $\mathcal{I}_A$ is transmitted to B. During the process, B learns $cert_A$ and the message msg. The protocol is executed successfully if msg can be verified under the public key of A and $cert_A$ can be verified under the public key of the authority.

2.2 Cryptographic Building Blocks

In order to ensure the security of our voting system we rely on two cryptographic primitives which we want to introduce in this section. These ingredients are functional encryption schemes and threshold blind signatures.

2.2.1 Threshold Blind Signature (TBS)

The notion of threshold blind signatures characterizes a combination of favourable properties of two cryptographic primitives. First blind signatures, which were proposed by Chaum [7] as a solution for the traceability problem in electronic payment systems. He showed, that the use of traditional digital signatures in aforementioned systems may break the anonymity of the payer, as the bank, which is responsible for signing the coins, is able to track the payer's purchase behaviour. A connection to the payer can be established, as soon as the bank receives an authenticated coin back from the merchant. With blind signatures this can be prevented, as in addition to the unforgeablitiy property of traditional digital signatures, blind signatures fulfils another blindness property. This ensures that the signer does not learn the true appearance of the signatures created by him. The second ingredient of threshold blind signatures are threshold signatures, which were introduced by Desmedt [12]. This primitive switches from a dedicated signer who creates the signatures, to a distributed signing process using secret sharing techniques. In order to create a valid signature for a message, t–out–of–n signers have to contribute partial signatures, which can later be combined. As this method moves the responsibility of the signing process to multiple independent entities, it comes useful in systems where reliability is a mandatory requirement.

Definition 1 (Threshold Blind Signature [24]) *A t-out-of-n threshold blind signature scheme* $\mathsf{TBS} = (\mathsf{TBS.ParGen}, \mathsf{TBS.KeyGen}, \mathsf{TBS.Sign}, \mathsf{TBS.Verify})$ *in a Common Reference String model consists of the following four algorithms:*

1. $\mathsf{TBS.ParGen}(1^n)$*: The* PPT *algorithm takes as input the security parameter 1^n and outputs public parameters I (possibly containing a common reference string crs_{TBS}).*
2. $\mathsf{TBS.KeyGen}(I)$*: On input public parameters I this algorithm outputs a secret share sk_i for each signer S_i, $i \in \{1, \ldots, n\}$ and a public key pk.*
3. $\mathsf{TBS.Sign}(\cdot)$*: This is a protocol between a user U and the signers S_i, $i \in \{1, \ldots, n\}$. The input of U is pk and a message m. The input of each server S_i is the secret share sk_i. The protocol results in a signature σ output by U.*
4. $\mathsf{TBS.Verify}(pk, m, \sigma)$*: A deterministic algorithm which on input a public key pk, message m, a signature σ outputs 1 if the signature is valid and 0 otherwise.*

We require that for every n, every I output by $\mathsf{TBS.ParGen}(1^n)$*, every set of secret keys sk_i and public key pk output by* $\mathsf{TBS.KeyGen}(I)$*, every signature σ output by* $\mathsf{TBS.Sign}(\cdot)$ *which is joint execution of a user $U(pk, m)$ with input the public key pk and an arbitrary message m from the appropriate underlying plaintext space, and a set of valid signers $S_i(sk_i)$ with input its secret share sk_i, it holds that* $\mathsf{TBS.Verify}(pk, m, \sigma) = 1$

For our system we require a secure t–out–of–n threshold blind signature scheme, which satisfies unforgeability and blindness properties. Informally, unforgeability means, that there exist a negligible success probability of an efficient adversary, which is able to corrupt at most

$t-1$ signers, of generating $k+1$ valid message-signatures pairs with different messages after at most k completed interactions with the honest set of signers. Further, blindness means that a malicious signer should not be able to distinguish, which of two messages m_0 and m_1 has been signed first, in two executions with an honest user, except with probability negligibly close to $\frac{1}{2}$. For an extension of the formal security definitions of Schröder et. al [37] to the threshold case, we refer to [24].

2.2.2 Decentralized Inner-Product Predicate Encryption (DIPPE)

In order to provide ballot privacy and trust within our voting system we rely on decentralized functional encryption. Functional encryption is a relativity new cryptographic primitive [36] that unfolds its full potential in applications that require data confidentiality and fine-grained access control. Unlike classical public-key encryption schemes, the output of the decryption algorithm of a functional encryption scheme is the result of a function on the input message. In case of an Inner-Product Predicate Encryption [29], the function is a predicate based on the inner-product of two vectors, a policy vector and an attribute vector. Within our system we make use of a decentralized version, as we aim to avoid a trusted "central" master key generating authority.

Definition 2 (Decentralized Inner-Product Predicate Encryption [29]) *A decentralized inner-product predicate encryption scheme is a tuple of probabilistic polynomial-time algorithms* $\mathsf{DIPPE} = (\mathsf{DIPPE.GlobalSetup}, \mathsf{DIPPE.AuthSetup}, \mathsf{DIPPE.KeyGen}, \mathsf{DIPPE.Encrypt}, \mathsf{DIPPE.Decrypt})$ *such that:*

1. $pp \leftarrow \mathsf{DIPPE.GlobalSetup}(1^n)$*: The probabilistic global setup algorithm takes as input the security parameter* 1^n *and outputs the master public parameters* pp

2. $\{sk_i, pk_i\} \leftarrow \mathsf{DIPPE.AuthSetup}(pp, i)$*: The probabilistic authority setup algorithm takes as input the public parameters* pp *and an authority identifier* i *from the authority universe. It outputs a pair* $\{sk_i, pk_i\}$ *consisting of the secret key* sk_i *and public key* pk_i *of authority* i*.*

3. $sk_{i,GID,\boldsymbol{v}} \leftarrow \mathsf{DIPPE.KeyGen}(pp, i, sk_i, \{pk_j\}_{j\neq i}, GID, \boldsymbol{v})$*: The probabilistic key generation algorithm takes as input the master public parameters* pp*, the authority index* i*, the secret key* sk_i *of the authority* i*, the public keys of all other authorities* $\{pk_j\}_{j\neq i}$*, a global identifier* GID *and the attribute vector* $\boldsymbol{v}$ *and outputs a secret* $sk_{i,GID,\boldsymbol{v}}$*.*

4. $ct \leftarrow \mathsf{DIPPE.Encrypt}(pp, \boldsymbol{x}, m)$*: The probabilistic encryption algorithm takes as input the master public parameters* pp*, the public keys of all authorities* $\{pk_i\}$*, the vector* $\boldsymbol{x}$ *which represents a policy* π *and a message* m *from the message space. The algorithm outputs a ciphertext* ct*.*

5. $m \leftarrow \mathsf{DIPPE.Decrypt}(ct, \{sk_{i,GID,\boldsymbol{v}}\}_{i=1}^{n})$*: The deterministic decryption algorithm takes as input the collection of obtained secret keys* $\{sk_{i,GID,\boldsymbol{v}}\}_{i=1}^{n}$*, and the ciphertext* ct *and outputs either the message* m *or the special symbol* $\perp$*.*

It is required that for every security parameter n, every master public parameter pp, every ciphertext policy vector $\boldsymbol{x}$, every attribute vector $\boldsymbol{v}$, and every secret $sk_{i,GID,\boldsymbol{v}}$, it holds that:

$$\mathsf{DIPPE.Decrypt}(\{sk_{i,GID,\boldsymbol{v}}\}_{i=1}^{n}, \mathsf{DIPPE.Encrypt}(pp, \boldsymbol{x}, m)) = \begin{cases} m & \textit{if } \langle \boldsymbol{x}, \boldsymbol{v} \rangle = 0 \\ \perp & \textit{else} \end{cases}$$

For our system we require that the Inner-Product Predicate Encryption scheme is semantically secure, which means informally, that a PPT adversary, who was given access to a key oracle which allows him to request decryption keys (that must not fulfil π_0 or π_1), has only negligible advantage in deciding which of either two messages m_0 and m_1 was encrypted by a challenger, that was given policies π_0, π_1 and a set of corrupted authorities. We refer to [29] for formal security definitions and examples of how to encode quorum policies as an inner product.

2.3 Hyperledger Fabric in a Nutshell

In this section we want to give an overview about Hyperledger Fabric[1], its components and how the flow to alter the ledger state looks like. Hyperledger Fabric is a distributed blockchain framework, which allows the integration of smart contracts, that utilizes Turing-complete programming languages. It is a permissioned ledger, which means, that there exist an additional access control layer to concretely define the tasks a specific network user is able to perform within the network. Hyperledger Fabric encompasses the following structures and participants:

Organisation: Organisations are superior entities within the context of Hyperleder Fabric. Every network participant, whether it is a client or a peer, is linked to an organisation through a digital certificate. A Hyperleder Fabric network, where multiple organisations work together, is called consortium.

Client: The client represents the end-user of a Hyperledger Fabric network. A client, equipped with a digital certificate issued by an Organization, is able to read the ledger or invoke a ledger change by interacting with a peer.

Peer: Within Hyperledger Fabric, there exist a flexible number of peers which act as the main entrance point for a client to communicate with the network in order to gain resources or to apply ledger changes. Every peer maintains an instance of the common ledger and provides a number of smart contracts which defines the way of how the ledger state can be changed. In order to initialize the collaboration, peers have to join a common channel. This channel is secure by design.

Ordering Service (Orderer): In Hyperledger Fabric there exist a dedicated ordering service which is responsible for the block creation process. While independent of existing

[1] `https://www.hyperledger.org/projects/fabric`

peers and channels, the orderer collects all transactions from clients and sorts them into blocks, following the first–come–first–serve principle. Provided options to implement the ordering service includes centralized solutions like Solo, but also decentralized ones like RAFT [33] or Kafka[2]. Apache Kafka is a distributed, scalable and crash-fault-tolerant messaging system, which implements a publish-subscribe pattern to organize access and processing of data. The setup encompasses data producers which feed their information into the Kafka cluster, which is a set of multiple nodes (Broker). The data itself is organized within topics and stored distributed within partitions on physical independent nodes. Data consumers are then able to subscribe to a topic in order to receive the data. In order to provide a coordination of the cluster, e.g. common lists of followers or leadership of specific partitions or topics, Kafka makes use of Zookeeper[3], which is a hierarchical key-value store with focus on high availability. It ensures that all Kafka nodes have access to the same set configuration data.

Hyperledger Transaction Flow. Now we briefly explain the process of altering the ledger state. We presume that all the cryptographic material has already been created and a common channel between attending peers is established. The purpose of channels in Hyperledger Fabric is to protect the transmitted data against external parties. They ensure data isolation and confidentiality for peers that work on a shared channel-specific ledger, since all parties must be properly authenticated to a channel in order to interact with it. Now, the program logic can be attached by installing and instantiating smart contracts on all peers within a channel. During the installation process the code is packed, signed and distributed to the peers. After the installation process, an authorized member (e.g. channel administrator) has to instantiate the smart contract by providing an *endorsement policy*. The policy defines which peers must endorse a request in order to be valid. This means, if a client wants to alter the ledger, he has to send a transaction proposal to all peers that are defined within the policy. On the other hand, the peers now simulate the request against the current ledger state and return a read-write set back to the client. The client then forwards the collected responses as a transaction message to the ordering service, which task is to order them chronologically and pack them into blocks. The created block is then broadcasted to all committing peers in the channel. They check that the endorsement policy is fulfilled and that there were no changes to the ledger state between the simulation step and this point. Each transaction within the block is now tagged as either valid or invalid. Finally, the block is attached to the ledger and each of the valid write sets are committed to the current state database.

3 System Model

In this section we introduce the system model of our voting application. The model comprises the entities of Hyperledger Fabric as described in Section 2.3 as well as *off-chain entities*

[2] https://kafka.apache.org/

[3] https://zookeeper.apache.org/

which offer the additional services required for the decentralization of the voting system.

Registrar: There exists a set of n_R registrars $\boldsymbol{R} := \{R_1, \dots, R_i, \dots, R_{n_R}\}$ which are privileged off-chain entities in the network. Every registrar R_i owns an identity $\mathcal{I}_{R_i} := \{sk_{R_i} pk_{R_i}, cert_{R_i}\}$ which consists of a secret key sk_{R_i}, a public key pk_{R_i} and further, a certificate $cert_{R_i}$ issued by an organisation of the network. The certificate contains a flag which labels R_i as a registrar. The main task of this entity is the registration of new votings within the system.

Voter: There exists a set of n_V off-chain voters $\boldsymbol{V} := \{V_1, \dots, V_i, \dots, V_{n_V}\}$, who intend to participate at a given voting. A voter V_i is in possession of two types of identities. First, an individual voter identity $\mathcal{I}_{V_i} := \{sk_{V_i}, pk_{V_i}, cert_{V_i}\}$ and second, an anonymous identity $\mathcal{I}_{anon} := \{sk_{anon}, pk_{anon}, cert_{anon}\}$ which is shared between all voters. Both identities contain valid certificates, issued from an organisation of the network, in combination with a pair of public and secret key. The purpose of the second anonymous identity is to ensure the privacy of the voter's ballot. Since we operate in a permissioned blockchain, every request has to be signed by an authorized member of the network. However, for our protocol we need the option to submit a ballot without revealing the real identity of the voter. By using this shared identity, a voter is able to bypass the membership requirement and to disguise the origin of the request. To participate at a given voting with id *vid*, voters have to retrieve all corresponding public keys $\{pk_{KA_i,vid}\}_{i \in \boldsymbol{x}}$, that are stored within the blockchain and cast an encrypted ballot out of it. The ballot $ballot_{vid,V_i}$ from a voter V_i is then submitted to the blockchain. Besides, voters are also allowed to fetch voting results from the blockchain.

Peer: There exists a set of n_P peers $\boldsymbol{P} := \{P_1, \dots, P_i \dots, P_{n_P}\}$. Every peer P_i with identity $\mathcal{I}_{P_i} := \{sk_{P_i}, pk_{P_i}, cert_{P_i}\}$. Main task of the peers is to maintain the distributed ledger. In addition, they provide the communication endpoint for all off-chain entities in the system. The application functionality is included within their smart contracts and can be invoked by appropriate entities. They offer interfaces for storing ballots and key material as well as to start the decryption and tallying processes.

Key Authority (Opening Oracle): There exists a set of n_{KA} off-chain key authorities $\boldsymbol{KA} := \{KA_1, \dots, KA_i, \dots, KA_{n_{KA}}\}$. Every key authority KA_i owns an identity $\mathcal{I}_{KA_i}$ $:= \{sk_{KA_i}, pk_{KA_i}, cert_{KA_i}\}$, issued by an organisation of the network. The certificate $cert_{KA_i}$ contains the index i together with a flag that labels the identity as a key authority. Key authorities fulfil two tasks within the voting system. First, for every registered voting, where KA_i is marked as a responsible key authority, KA_i creates a new key pair $(pk_{KA_i,vid}, sk_{KA_i,vid})$ consisting of public and private key. The public $pk_{KA_i,vid}$ is then published to the blockchain. The second task relates to the opening process. In order to dissolve a voting, every responsible key authority KA_i has to generate a decryption key $sk_{KA_i,vid,\boldsymbol{v}}$ using its secret key $sk_{KA_i,vid}$ and a policy vector $\boldsymbol{v}$, and publishes it to the blockchain.

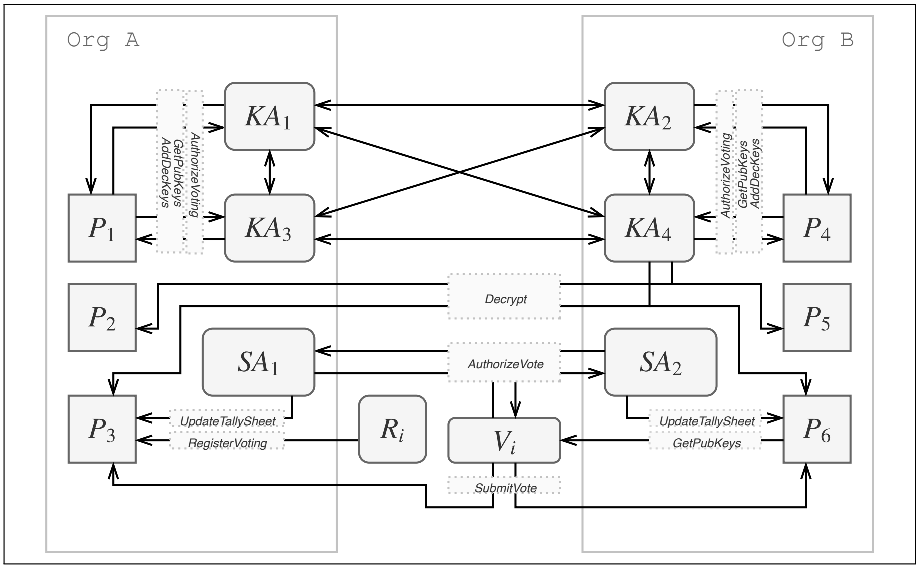


Figure 1: Overview of our decentralized voting system. (For presentation purposes the orderer is not shown)

Signer Authority (Anonymizer Oracle): There exists a set of n_{SA} off-chain signer authorities $\boldsymbol{SA} := \{SA_1, \ldots, SA_i, \ldots, SA_{n_{SA}}\}$. Every signer authority SA_i with identity $\mathcal{I}_{SA_i} := \{sk_{SA_i}, pk_{SA_i}, cert_{SA_i}\}$. Like key authorities, a signer authority owns a special attribute within its certificate $cert_{SA_i}$ that labels itself as a signer authority in the network. The task of the signer authority SA_i is to create a signature σ_{vid,V_i} for a given ballot of voter V_i. The signature enables a voter to place his ballot within the blockchain. This step is necessary in order to prevent a double voting of the voter by utilizing his anonymous identity.

Ordering Service (Orderer): There exists a set of n_O orderers $\boldsymbol{O} = \{O_1, \ldots, O_{n_O}\}$ which main responsibility is to take part at the consensus and block creation process.

Organisation: There exists a set of n_{Org} organisations $\boldsymbol{Org} = \{Org_1, \ldots, Org_i, \ldots, Org_{n_{Org}}\}$ with key pairs (pk_{Org_i}, sk_{Org_i}) consisting of one public and one private key. Entities are able to obtain an organisation membership by requesting a certificate, that is signed with the secret key of the organisation.

4 Protocol

In this section we explain the structure of our voting system. An overview is depicted in Fig. 1. Without loss of generality we assume a setting encompassing two organisations, called Org_A and Org_B. Every organisation contributes with peers, registrars, signer and key authorities to the network. The voting system consists of four protocols: *Setup*, *Pre-Voting*, *Voting* and *Tallying*.

4.1 Setup

The task of the setup protocol is to initialize a ballot and setup the entities entitled to participate in the voting. We assume that the blockchain not only serves as a provider of the voting platform. Its storage capabilities are also used to share publicly available cryptographic key material and infrastructural information for off-chain entities.

4.1.1 Preparation of the Hyperledger Fabric Network

The first step during the setup phase is the preparation of the Hyperledger Fabric network. We only give a brief summary here, as this is a standard process and beyond the scope of this paper. For more details, we refer to the Hyperledger Fabric documentation [21]. First, each participating organisation has to generate their cryptographic material. This includes keys for the organisation itself and also for peers, which act on behalf of the organisation. Now, a member of the consortium uses all public available material to create a common configuration (genesis block and channel description) which is necessary in order to boot network components and to establish a secure communication channel. Note, that it is also possible to add organisations afterwards. The next step is the installation and initialization of the voting smart contract on all participating peers. This step also includes the definition of an endorsement policy that defines the conditions under which a change of the ledger state is enforced.

4.1.2 Preparation of off-chain Entities

All registrars, signer authorities and key authorities now generate a new key pair consisting of a public and private key. In order to get the authorisation to perform operations within the network, they have to obtain a certificate, signed with the secret key of an organisation. Each certificate contains a flag, which defines the entity authorization type. The purpose is to limit the operation set that the entity is able to perform within the network. To obtain a digital certificate voters need to prove their physical identity to an organisation. Details of the issuance policy as well as the decentralized implementation of the certification protocol (e.g. through an MPC protocol [9]) are out of scope.

4.1.3 Registration of Signer Authorities

As we focus on a complete decentralized voting platform, the signer authorities start by engaging the common execution of the $\mathsf{TBS.KeyGen}(I_{\mathsf{TBS}})$ algorithm using the pre-generated public parameters I_{TBS}, which relies on some distributed key generation protocol with output the set of secret shares sk_{TBS,SA_i} for each signer authority SA_i and public key components that form the common public key pk_{TBS}. The public key components are published by sending a signed message to the blockchain, which on the other hand verifies that the signer authority is indeed authorized to perform this action. In order to prevent double voting, the blockchain accepts a ballot, only if the ballot contains a signature verifiable under the public key pk_{TBS}. While a valid signature can only be obtained once per voter V_i and voting id vid, the signer authorities keep track of already issued partial blind signatures within their local tally sheet. The entire protocol for registering a new set of signer authorities is given in Fig. 2.

$\mathbf{SignerAuthority}_i(I_{\mathsf{TBS}}, \mathcal{I}_{SA_i})$		$\mathbf{Peer}_j(\cdot)$
$(sk_{\mathsf{TBS},SA_i}, pk_{\mathsf{TBS}}) \leftarrow \mathsf{TBS.KeyGen}(I_{\mathsf{TBS}})$		
	$\mathcal{I}_{SA_i} \xrightarrow{pk_{\mathsf{TBS}}}$	
		$b := \mathsf{CheckSignerPermission}(cert_{SA_i})$ **if** $b = \mathbf{ok}$ $\quad\mathsf{RegisterDistributedSigner}(pk_{\mathsf{TBS}})$

Figure 2: The sub-protocol for registering a new set of signer authorities

4.2 Pre-Voting

Within the pre-voting phase, a new voting is created by a registrar and authorized by a set of responsible key authorities.

4.2.1 Voting Registration

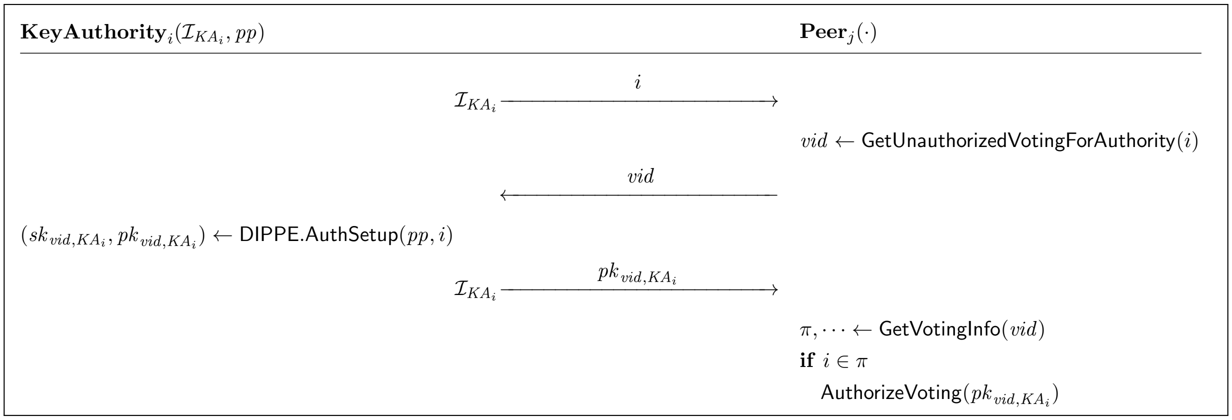
To schedule a new voting within the system (Fig. 3), a registrar R_i sends a signed request comprising the voting id *vid*, a name *name*, a set of eligible voters $\boldsymbol{ev}$, a set $\boldsymbol{opt}$ of possible choices and a policy π, which specifies a set of responsible key authorities to the blockchain. The blockchain on the other hand verifies the authorization of R_i and in case of success registers the new voting within in the system.

$\mathbf{Registrar}_i(\mathcal{I}_{R_i})$		$\mathbf{Peer}_j(\cdot)$
	$\mathcal{I}_{R_i} \xrightarrow{vid,\, name,\, \boldsymbol{ev},\, \boldsymbol{opt},\, \pi}$	
		$b := \mathsf{CheckRegistrarPermission}(cert_{R_i})$ **if** $b = \mathbf{ok}$ $\quad\mathsf{RegisterVoting}(vid, \boldsymbol{opt}, \pi, name, \boldsymbol{ev})$

Figure 3: The sub-protocol for registering a new voting

4.2.2 Voting Authorisation

Before voters can participate in the registered voting, the voting needs to be authorized by all key authorities included within the policy π (Fig. 4). A key authority KA_i therefore fetches the *vid* of the registered voting from the blockchain and creates a new functional encryption key pair $\{sk_{vid,i}, pk_{vid,KA_i}\} \leftarrow \mathsf{DIPPE.AuthSetup}(pp, KA_i)$ using the public parameters pp and its identifier i from the key authority universe. While the secret key sk_{vid,KA_i} is stored locally, the public key pk_{vid,KA_i} is published in a signed request to a blockchain peer. If the blockchain is able to verify that the requester is part of the voting policy, then it accepts and attaches the public key to the voting object which is stored within the blockchain.



$\mathbf{KeyAuthority}_i(\mathcal{I}_{KA_i}, pp)$
$\mathbf{Peer}_j(\cdot)$
$\mathcal{I}_{KA_i} \xrightarrow{i}$
$vid \leftarrow \mathsf{GetUnauthorizedVotingForAuthority}(i)$
$\xleftarrow{vid}$
$(sk_{vid,KA_i}, pk_{vid,KA_i}) \leftarrow \mathsf{DIPPE.AuthSetup}(pp, i)$
$\mathcal{I}_{KA_i} \xrightarrow{pk_{vid,KA_i}}$
$\pi, \cdots \leftarrow \mathsf{GetVotingInfo}(vid)$
if $i \in \pi$
$\mathsf{AuthorizeVoting}(pk_{vid,KA_i})$

Figure 4: The sub-protocol for authorizing a registered voting

4.3 Voting

In the voting phase, a ballot is casted locally by the voter and accredited by a set of signer authorities, before it is submitted to the blockchain network.

4.3.1 Ballot Casting and Accreditation

In order to cast a new ballot for a voting *vid* (Fig. 5), the voter first has to retrieve all required parameters like the collection of available options $\boldsymbol{opt}$, the policy π and the set of public keys $\{pk_{KA_m,vid}\}_{m\in\pi}$, from the blockchain. Now V_i calls $ballot_{vid,V_i} \leftarrow \mathsf{DIPPE.Encrypt}(pp, \boldsymbol{x}, opt_{vid,V_i})$ using his choice $opt_{vid,V_i} \in \boldsymbol{opt}$, the associated public keys $\{pk_{KA_m,vid}\}_{m\in\pi}$, the policy vector $\boldsymbol{x}$ derived from the policy π and the scheme's public parameter pp. In order to prevent double voting, the ballot $ballot_{vid,V_i}$ has to be signed by a set of at least t signing authorities SA_j. Therefore, they check whether the voter V_i is eligible to participate at the voting *vid* and if that is the case, then V_i with input his ballot $ballot_{vid,V_i}$ and the set of signer authorities engages in the protocol $\mathsf{TBS.Sign}(\cdot)$, while in the end V_i obtains a signature σ_{vid,V_i} and all participating SA_j learn nothing about the ballot. Each participating signer authority now adds the identifier i of the voter to their local tally sheet.

4.3.2 Ballot Submission

V_i now uses his anonymous identity I_{anon} to place the ballot $ballot_{vid,V_i}$ together with the signature σ_{vid,V_i} within the blockchain. Because this is done anonymously, the blockchain learns nothing about the identity of V_i. The blockchain accepts the vote if $\mathsf{TBS.Verify}(pk_{\mathsf{TBS}}, ballot_{vid,V_i}, \sigma_{vid,V_i})$ outputs 1. The protocol for submitting the ballot to the blockchain is detailed in Fig. 6.

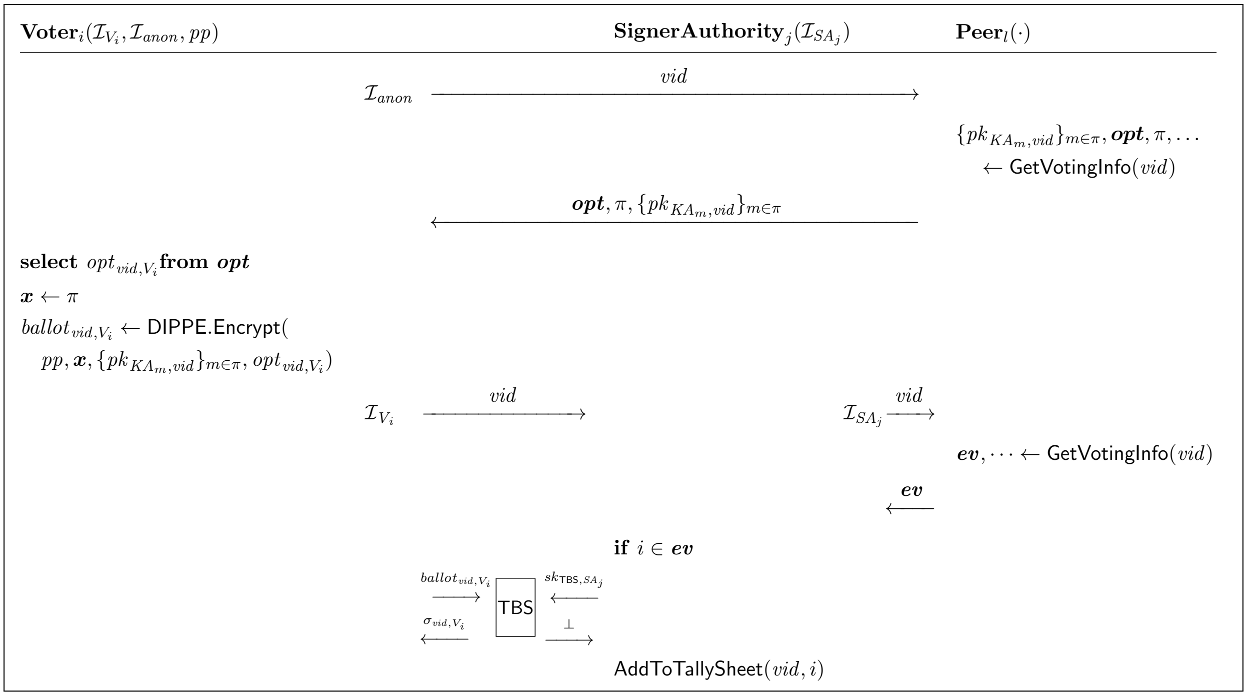

$\mathbf{Voter}_i(\mathcal{I}_{V_i}, \mathcal{I}_{anon}, pp)$
$\mathbf{SignerAuthority}_j(\mathcal{I}_{SA_j})$
$\mathbf{Peer}_l(\cdot)$
$\mathcal{I}_{anon}$
vid
$\{pk_{KA_m,vid}\}_{m\in\pi}, \boldsymbol{opt}, \pi, \ldots$
$\leftarrow \mathsf{GetVotingInfo}(vid)$
$\boldsymbol{opt}, \pi, \{pk_{KA_m,vid}\}_{m\in\pi}$
$\textbf{select } opt_{vid,V_i} \textbf{from } \boldsymbol{opt}$
$\boldsymbol{x} \leftarrow \pi$
$ballot_{vid,V_i} \leftarrow \mathsf{DIPPE.Encrypt}($
$pp, \boldsymbol{x}, \{pk_{KA_m,vid}\}_{m\in\pi}, opt_{vid,V_i})$
$\mathcal{I}_{V_i}$
vid
$\mathcal{I}_{SA_j}$
vid
$\boldsymbol{ev}, \cdots \leftarrow \mathsf{GetVotingInfo}(vid)$
$\boldsymbol{ev}$
$\textbf{if } i \in \boldsymbol{ev}$
$ballot_{vid,V_i}$
sk_{TBS,SA_j}
TBS
σ_{vid,V_i}
$\perp$
$\mathsf{AddToTallySheet}(vid, i)$

Figure 5: The sub-protocol for casting a new ballot

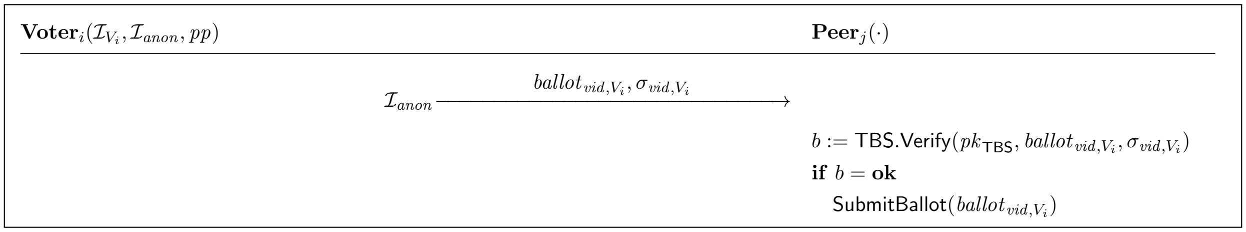

$\mathbf{Voter}_i(\mathcal{I}_{V_i}, \mathcal{I}_{anon}, pp)$
$\mathbf{Peer}_j(\cdot)$
$\mathcal{I}_{anon}$
$ballot_{vid,V_i}, \sigma_{vid,V_i}$
$b := \mathsf{TBS.Verify}(pk_{\mathsf{TBS}}, ballot_{vid,V_i}, \sigma_{vid,V_i})$
$\textbf{if } b = \textbf{ok}$
$\mathsf{SubmitBallot}(ballot_{vid,V_i})$

Figure 6: The sub-protocol for submitting a ballot to the blockchain

4.4 Tallying

In the tallying phase, encrypted ballots are opened with the help of decryption keys provided by the key authorities.

4.4.1 Unlocking Votes

If there exists a voting vid that is about to expire, all responsible key authorities KA_i request the set of public keys attached to the voting object, together with the policy π from the blockchain (Fig. 7). With help of these parameters, they generate a decryption key $sk_{KA_i,vid,\boldsymbol{v}} \leftarrow \mathsf{DIPPE.KeyGen}(pp, i, sk_{KA_i,vid}, \{pk_{KA_m,vid}\}_{m\in\pi}, vid, \boldsymbol{v})$ using their own secret key sk_{vid,KA_i}, the public keys $\{pk_{KA_m,vid}\}_{m\in\pi}$ of all responsible key authorities that are stored within the blockchain, and an attribute vector $\boldsymbol{v}$ that derives from the policy π. The decryption key $sk_{KA_i,vid,\boldsymbol{v}}$ is then published to the blockchain using a signed request and the blockchain on the other hand verifies that the key authority KA_i is indeed responsible for this voting. If that is the case, then the blockchain attaches the decryption key to the voting object stored within the blockchain.

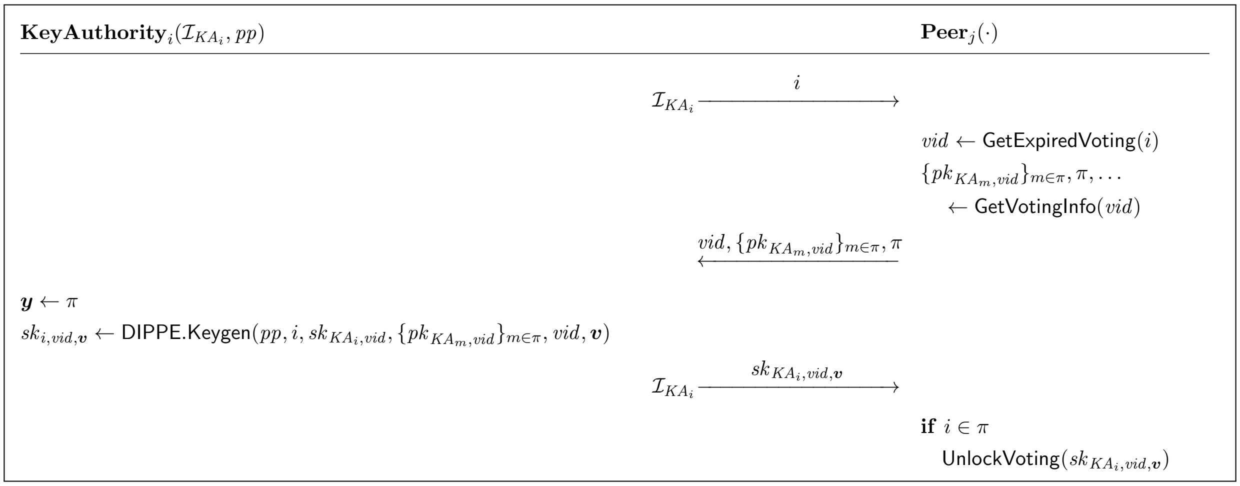

$\textbf{KeyAuthority}_i(\mathcal{I}_{KA_i}, pp)$
$\textbf{Peer}_j(\cdot)$
$\mathcal{I}_{KA_i} \xrightarrow{i}$
$vid \leftarrow \mathsf{GetExpiredVoting}(i)$
$\{pk_{KA_m,vid}\}_{m\in\pi}, \pi, \ldots$
$\leftarrow \mathsf{GetVotingInfo}(vid)$
$\xleftarrow{vid, \{pk_{KA_m,vid}\}_{m\in\pi}, \pi}$
$\boldsymbol{y} \leftarrow \pi$
$sk_{i,vid,\boldsymbol{v}} \leftarrow \mathsf{DIPPE.Keygen}(pp, i, sk_{KA_i,vid}, \{pk_{KA_m,vid}\}_{m\in\pi}, vid, \boldsymbol{v})$
$\mathcal{I}_{KA_i} \xrightarrow{sk_{KA_i,vid,\boldsymbol{v}}}$
$\textbf{if } i \in \pi$
$\mathsf{UnlockVoting}(sk_{KA_i,vid,\boldsymbol{v}})$

Figure 7: The sub-protocol for creating a new decryption key

4.4.2 Finalize Voting

The actual decryption process (Fig. 8) is triggered by one of the responsible key authorities KA_i. The decryption of the stored votes occurs within the smart contract as a fully transparent process. Therefore, the choice opt_{vid,V_m} of voter V_m is revealed by calculating $opt_{vid,V_m} := \mathsf{DIPPE.Decrypt}(ballot_{vid,V_m}, \{sk_{KA_i,vid,\boldsymbol{v}}\})$ and stored within the blockchain. Voters are now able to request the voting result from the blockchain.

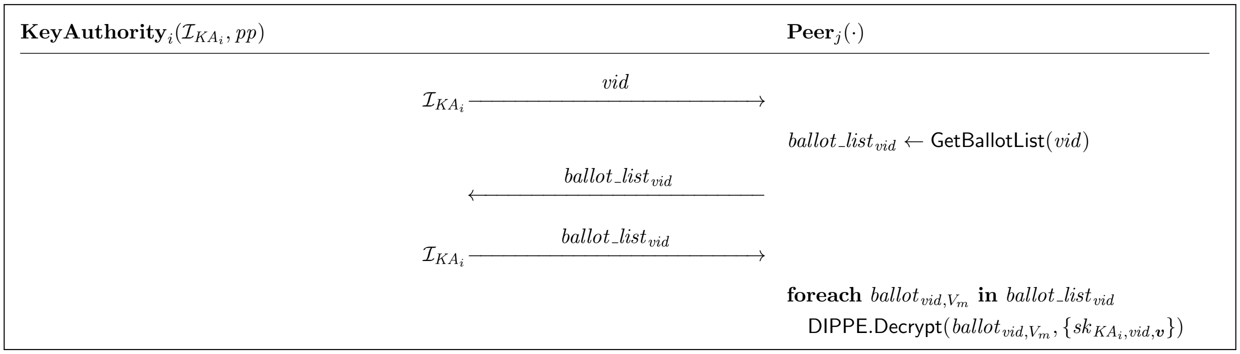

$\textbf{KeyAuthority}_i(\mathcal{I}_{KA_i}, pp)$
$\textbf{Peer}_j(\cdot)$
$\mathcal{I}_{KA_i} \xrightarrow{vid}$
$ballot\_list_{vid} \leftarrow \mathsf{GetBallotList}(vid)$
$\xleftarrow{ballot\_list_{vid}}$
$\mathcal{I}_{KA_i} \xrightarrow{ballot\_list_{vid}}$
$\textbf{foreach } ballot_{vid,V_m} \textbf{ in } ballot\_list_{vid}$
$\mathsf{DIPPE.Decrypt}(ballot_{vid,V_m}, \{sk_{KA_i,vid,\boldsymbol{v}}\})$

Figure 8: The sub-protocol for opening the encrypted ballots

5 Security Analysis

In this section, we analyze and informally argue security of our voting system. Therefore, we assume that the underlying Hyperledger Fabric network is correctly configured, that critical key material of the individual members is kept private and the communication channels between the different network members are TLS-secured.

Ballot Privacy: One of the most fundamental security properties of a voting system is ballot privacy. It literally means that an adversary is not able to reveal the way a specific voter has voted. Our system ensures ballot privacy due to the fact that voters submit their ballots using a distinct anonymous identity, which is shared between all voters and therefore disguises the real user behind it. Even if ballots that are stored within the blockchain become publicly visible, a connection to a specific voter can not be established any more. In order to prevent double voting, a ballot has to be submitted together with a signature over the ballot, created by a set of signer authorities. The blindness property of the threshold blind signature scheme ensures, that even a malicious set of signer authorities is not able to learn the structure of the ballot and as a consequence break the privacy as soon as the ballot is published to the blockchain.

Fairness: Fairness means that an adversary is not able to obtain intermediate results of a voting before its expiration date. This property is ensured within our system as long as no majority of malicious key authorities collude. The reason is that all ballots which enter the system are encrypted client-side using an IND-CPA secure functional encryption scheme and in order to open ballots that are stored within the blockchain, decryption keys from multiple key authorities are required.

Eligibility: The eligibility property limits the voting attendance to voters that are entitled to it. Each voter within our system possesses a digital identity which they obtain in exchange to a proof of their real identity. When registering a new voting at the system, a registrar is able to define the set of voters who are able to participate. The enforcement of the eligibility property is then performed by the signer authorities. They check whether the voting description within the blockchain matches against the certificate of the requesting voter. In case the voter is eligible, they continue and create a signature, else they decline.

Verifiability: Verifiability means that a voter is able to validate the correctness of the system. Therefore, we analyse two different segments of the protocol. Note, that within the permissioned blockchain setting, the ledger is stored on the peers, and therefore beyond the direct access of the voters. Our way to counteract this disadvantage is by using multiple independent organisations which hosts the network peers. We require that each organisation provides the blockchain data to the public. In case one organisation publishes false results it will be noticed and cheating organisations can be expelled.

Cast-as-intended verifiability: This property makes sure, that a ballot was not altered during its generation. It holds within our system because the ballot is casted client-side.

Recorded-as-cast verifiability: A voter is able to verify that his casted ballot was recorded as intended. This is the case, because the set of encrypted ballots is publicly accessible within the blockchain. Any voter can verify that his casted ballot is the same as within the blockchain.

Tallied-as-recorded verifiability: Another important property includes the ability to verify that the counting process was executed honestly by the voting system. This is ensured as the voter gains access to the blockchain, which contains all ballots and post-opening all associated choices in cleartext. Each voter is able to manually recount the published results.

Double-voting resistance: Double-voting is the threat in 1–person–1–vote systems, that a voter is able to submit more than just one ballot. In our system, double-voting is prevented as long as a majority of signer authorities acts honestly. This is the case because a ballot can only be submitted in combination with a valid signature, issued by a set of signer authorities. Signer authorities however will only issue signatures once for every eligible voter. Further we require that the used threshold blind signature scheme fulfils the unforgeable property, which prevents that valid signatures can be unwarrantably casted by unauthorized entities.

Censorship resistance: Our voting system is censor resistant as long as there exists at least one honest organisation (voting provider). This prevents harmful modifications to the voting program and also the undetected addition of forgeries or the removal of submitted ballots from the distributed database. This is the case because the blockchain itself is immutable and changes can be detected by inspecting the blockchain data structure.

Reliability: The fact that our voting system is build on top of a decentralized blockchain ensures protection against data loss. A dedicated copy of the database is stored on each peer and even if there is a malfunction on some of them, the system remains functional.

A problem we do not analyze further in this work is the potential susceptibility to side channel attacks. For example, through running a timing attack the adversary may deduce the temporal relation between the moment a voter requests a signature for the ballot and the moment a voter submits the signed ballot to the blockchain. Voters may become as well traceable due to the design of the Internet and the way IP addresses are distributed. While the first type of timing attacks can be avoided by adding on purpose random delays between ballot casting and submission, the second type may be resolved by tunnelling the communication through anonymisation networks like TOR[4][7].

[4] `https://www.torproject.org/`

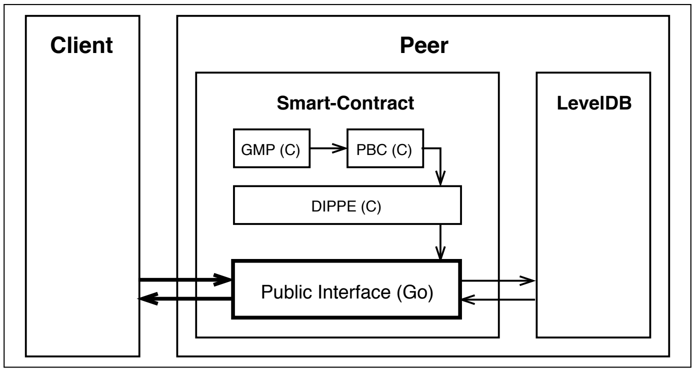


Figure 9: Structure of the smart contract (chain code) within a peer

6 Implementation

In this section we report on the performance evaluation of our reference implementation in the Hyperledger Fabric framework. We also discuss optimization techniques in the smart contract to scale the performance.

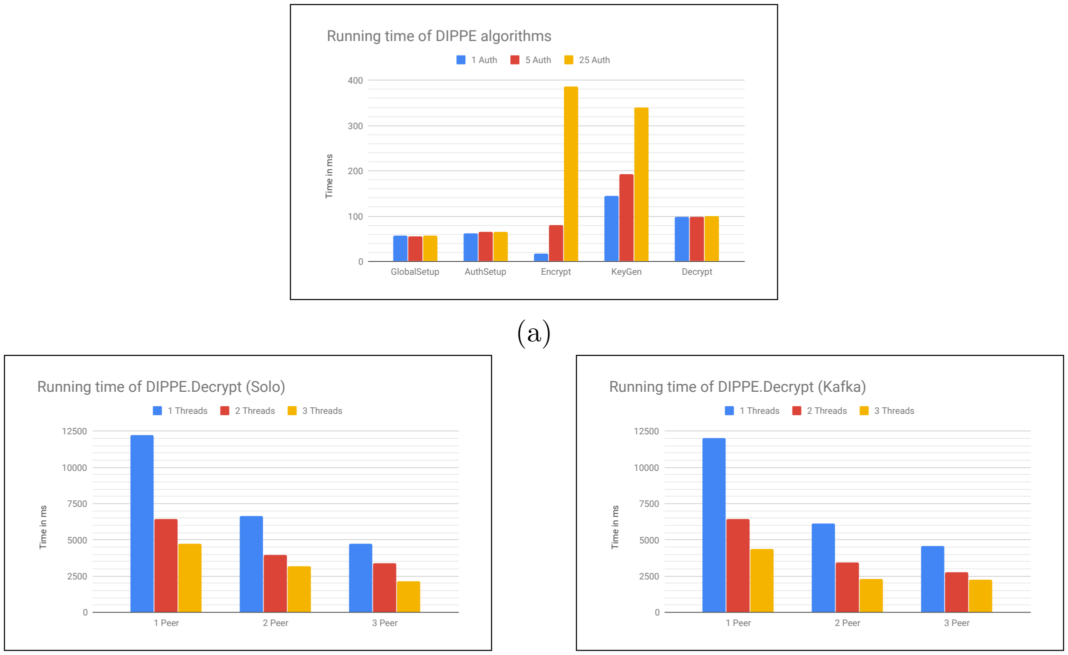
6.1 Setup

The prototype of the voting system runs within the framework of Hyperledger Fabric version 1.2. Our setup encompasses a Proxmox[5] server cluster facilitating the virtualization of each component of the Hyperledger Fabric network (peers, zookeeper- and kafka-nodes) in a single virtual machine (VM). Each VM allocates a single core of the host's Intel Core i7-4790 CPU @ 3,60 GHz with 512 MB RAM running the small and resource-saving Alpine Linux 3.8 operating system. In addition, the Hyperledger Fabric components run within separated Docker containers, as recommended for production use. The Docker runs in swarm mode, which is a network overlay that connects multiple Docker daemons together and enables cross-communication among containers between different VMs. The overlay sits on top of the host-specific network and transparently handles the routing of each packet to and from the Docker daemon host and the destination container. Each peer of the Hyperledger Fabric network maintains a copy of the ledger stored in a LevelDB[6] database. LevelDB is a simple and fast key-value store. Simple in this context means that solely a single string is written and read from the key-value store, respectively. This simplicity comes with the lack of a rich query language like for example SQL.

Our implementation of the voting system's actual program logic, which exists on the peers in form of a smart contract, is carried out in Golang. We have chosen Golang because of its compatibility with the programming language C. The exchange of data between the Go and C parts of our smart contract is enabled by the pseudo-package "C". It enables the shared access on variables and also the call of functions from Go in C and vice versa

[5] https://www.proxmox.com/

[6] https://github.com/google/leveldb



(a)

(b)

(c)

Figure 10: (a) Time comparison of algorithms of the DIPPE scheme. (Other operations add negligible computational overhead to the voting system and are left out.) The underlying curve is MNT224 and the DLIN assumption size is 2. (b-c) Performance improvements due to multi-threading and clustering with a decentralized Kafka orderer with 4 Nodes vs. a Solo orderer (100 votes, single-authority policy)

(Fig. 9). A pivotal part of the smart contract is devoted to execute the algorithms of the Decentralized Inner-Product Predicate Encryption scheme. The decryption algorithm of [29] specifically makes use of a pairing function. We note that the other cryptographic operations in the implementation of the voting system, e.g. (threshold) signature generation and verification, are fairly standard and have negligible computational costs in comparison to the pairing operations of the DIPPE scheme. They are also invoked once per voter and vote authentication. Hence they are left out from the discussion. We instantiate the DIPPE with the construction of [29] over MNT224 elliptic curves and the DLIN assumption size 2. The arithmetic operations are taken from the Pairing-Based Cryptography Library[7] (PBC) which provides the pairing operation and the GNU Multiple Precision Arithmetic Library[8] (GMP), which provides the field operations.

6.2 Performance Evaluation

Unlike other blockchain frameworks, which store the application code as byte code within the ledger and later execute it within a virtual machine, Hyperledger Fabric chain code must be installed manually on a peer. This approach results in an executable machine code that is compiled for this target system, capitalises on further optimisations and reaches nearly optimal performance.

In order to tally a ballot, it is necessary to decrypt all encrypted votes stored in the blockchain. If the smart contract naively decrypts the votes one–by–one, the tally time complexity is linear in the number of ballots. For large quantities, say 10^7, the time would amount to approximately 340 hours using a single-threaded peer. We leverage two optimization techniques in the smart contract, namely multi-threading and clustering.

Multi-Threading: We make use of multi-threading to reduce the time required for decryption. We gain the greatest benefit out of this method if the machines running the chain code support multiple cores.

Peer Clustering: A more flexible method offers clustering. Instead of allowing every peer to decrypt the whole set of votes within the chain code, we ask distinguished peers to decrypt only a subset of the encrypted votes. Consider, for example, a consortium consisting of two organisations each providing two peer nodes. Now instead of asking for the decryption of the encrypted votes by all peer nodes, divide the number of decryptions and ask one node from every organization to do the decryption. While the idea is simple, it approximately halves the decryption time (see Fig. 10c).

A summary of the running time for the critical algorithms of the DIPPE scheme and the decryption time optimized with multi-threading and sharding is illustrated in Fig. 10a and 10b-10c, respectively. The performance measurement in Fig. 10a shows that the time of DIPPE.Encrypt and DIPPE.KeyGen scales with the number of key authorities. This is uncritical because DIPPE.Encrypt is invoked once for every voter and ballot *off-chain*. A similar argument applies for the key authority calling DIPPE.KeyGen. Interestingly, the number of key authorities has little impact on the time of the DIPPE.Decrypt-algorithm and thus the size of the decryption policy, making [29] a good choice for decentralizing trust in opening the votes. Also scaling the trustlessness of the hyperledger blockchain network from semi-decentralization (Solo ordering service, Fig. 10b) to full decentralization (Kafka ordering service, Fig. 10c) has marginal effects on the overall performance.

7 Conclusion

In this paper we propose a trustless, censorship-resilient and scaleable electronic voting system based on Hyperledger Fabric. Our system allows the parameterisation of the number

[7] `https://crypto.stanford.edu/pbc/`

[8] `https://gmplib.org/`

of Hyperledger Fabric peers in order to decrease the required tallying time of the ballots. Thus, when considering large-scale votings like for example the federal election 2017 in Germany with around 61,69 million eligible voters, our voting system approximately needs 512 single-threaded peers for each organisation in order to perform the whole counting process in around 8,77 hours, which is within the time frame of traditional, manual tallying processes. Using multi-threading may enhance the efficiency further more. The same holds for faster functional encryption schemes, which are subject to on-going research and may be available in the future.

References

[1] E. Androulaki et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains". In: *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018.* 2018, 30:1–30:15.

[2] S. I. András. *Implementing an e-voting protocol with blind signatures on Ethereum.* 2018. URL: `http://medium.com/coinmonks/implementing-an-e-voting-protocol-with-blind-signatures-on-ethereum-411e88af044a` (visited on 03/13/2019).

[3] V. P. Binu, D. G. Nair, and A. Sreekumar. "Secret Sharing Homomorphism and Secure E-voting". In: *CoRR* abs/1602.05372 (2016). arXiv: `1602.05372`. URL: `http://arxiv.org/abs/1602.05372`.

[4] V. Brusco, M. Nazareno, and S. C. Stokes. "Vote buying in Argentina". In: *Latin American Research Review* 39.2 (2004), pp. 66–88.

[5] C. Cachin and M. Vukolic. "Blockchain Consensus Protocols in the Wild". In: *CoRR* abs/1707.01873 (2017). arXiv: `1707.01873`. URL: `http://arxiv.org/abs/1707.01873`.

[6] J. Camenisch and E. V. Herreweghen. "Design and implementation of the *idemix* anonymous credential system". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002.* 2002, pp. 21–30.

[7] D. Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Commun. ACM* 24.2 (1981), pp. 84–88.

[8] X. Cheng, W. Xu, and X. Wang. "A threshold blind signature from well pairing on elliptic curves". In: *Journal of Electronics (China)* 23.1 (2006), pp. 76–80. ISSN: 1993-0615.

[9] K. Chida et al. "Fast Large-Scale Honest-Majority MPC for Malicious Adversaries". In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III.* 2018, pp. 34–64.

[10] D. N. Committee et al. *Democracy at Risk: The 2004 Election in Ohio.* 2005.

[11] DAOstack. *DAOstack - An operating system for collective intelligence.* URL: https://daostack.io/ (visited on 04/19/2019).

[12] Y. Desmedt. "Society and Group Oriented Cryptography: A New Concept". In: *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings.* 1987, pp. 120–127.

[13] A. J. Feldman, J. A. Halderman, and E. W. Felten. "Security Analysis of the Diebold AccuVote-TS Voting Machine". In: *2007 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'07, Boston, MA, USA, August 6, 2007.* 2007.

[14] Freedom House. *Freedom in the World 2019.* 2019. URL: https://freedomhouse.org/sites/default/files/Feb2019_FH_FITW_2019_Report_ForWeb-compressed.pdf (visited on 04/15/2019).

[15] S. F. Freeman. "The unexplained exit poll discrepancy". In: *University of Pennsylvania Graduate School of Arts and Sciences, Center for Organizational Dynamics Research Report* (2004), pp. 04–09.

[16] A. Fujioka, T. Okamoto, and K. Ohta. "A Practical Secret Voting Scheme for Large Scale Elections". In: *Advances in Cryptology - AUSCRYPT '92, Workshop on the Theory and Application of Cryptographic Techniques, Gold Coast, Queensland, Australia, December 13-16, 1992, Proceedings.* 1992, pp. 244–251.

[17] S. Gajek. *2019 is the year of DAOs – Now we urgently need robust Consensus protocols for the People.* 2019. URL: https://hackernoon.com/2019-is-the-year-of-daos-9728618873f5 (visited on 04/21/2019).

[18] F. S. Hardwick, R. N. Akram, and K. Markantonakis. "E-Voting with Blockchain: An E-Voting Protocol with Decentralisation and Voter Privacy". In: *CoRR* abs/1805.10258 (2018). arXiv: 1805.10258. URL: http://arxiv.org/abs/1805.10258.

[19] S. Heiberg et al. "On Trade-offs of Applying Block Chains for Electronic Voting Bulletin Boards". In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 685. URL: https://eprint.iacr.org/2018/685.

[20] M. Hirt and K. Sako. "Efficient Receipt-Free Voting Based on Homomorphic Encryption". In: *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding.* 2000, pp. 539–556.

[21] Hyperledger. *A Blockchain Platform for the Enterprise - Hyperledger Fabric.* 2019. URL: https://hyperledger-fabric.readthedocs.io/ (visited on 03/21/2019).

[22] W. Juang and C. Lei. "Blind Threshold Signatures Based on Discrete Logarithm". In: *Concurrency and Parallelism, Programming, Networking, and Security: Second Asian Computing Science Conference, ASIAN '96, Singapore, December 2-5, 1996, Proceedings.* 1996, pp. 172–181.

[23] J. Katz, A. Sahai, and B. Waters. "Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products". In: *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings.* 2008, pp. 146–162.

[24] V. Kuchta and M. Manulis. "Rerandomizable Threshold Blind Signatures". In: *Trusted Systems - 6th International Conference, INTRUST 2014, Beijing, China, December 16-17, 2014, Revised Selected Papers.* 2014, pp. 70–89.

[25] C. Lei, W. Juang, and P. Yu. "Provably Secure Blind Threshold Signatures Based on Discrete Logarithm". In: *J. Inf. Sci. Eng.* 18.1 (2002), pp. 23–39.

[26] Y. Liu and Q. Wang. "An E-voting Protocol Based on Blockchain". In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 1043. URL: `http://eprint.iacr.org/2017/1043`.

[27] P. McCorry, S. F. Shahandashti, and F. Hao. "A Smart Contract for Boardroom Voting with Maximum Voter Privacy". In: *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers.* 2017, pp. 357–375.

[28] W. Mebane and K. Kalinin. *Comparative Election Fraud Detection.* 2009.

[29] Y. Michalevsky and M. Joye. "Decentralized Policy-Hiding Attribute-Based Encryption with Receiver Privacy". In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 753. URL: `https://eprint.iacr.org/2018/753`.

[30] Y. Nasser et al. *Blockchains and Voting: Somewhere between hype and a panacea (A Position Paper).* 2018.

[31] W. Ogata et al. "Fault tolerant anonymous channel". In: *Information and Communication Security, First International Conference, ICICS'97, Beijing, China, November 11-14, 1997, Proceedings.* 1997, pp. 440–444.

[32] T. Okamoto. "Receipt-Free Electronic Voting Schemes for Large Scale Elections". In: *Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings.* 1997, pp. 25–35.

[33] D. Ongaro and J. K. Ousterhout. "In Search of an Understandable Consensus Algorithm". In: *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014.* 2014, pp. 305–319.

[34] C. Park, K. Itoh, and K. Kurosawa. "Efficient Anonymous Channel and All/Nothing Election Scheme". In: *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings.* 1993, pp. 248–259.

[35] K. Peng et al. "Multiplicative Homomorphic E-Voting". In: *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings.* 2004, pp. 61–72.

[36] A. Sahai and B. Waters. "Fuzzy Identity-Based Encryption". In: *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*. 2005, pp. 457–473.

[37] D. Schröder and D. Unruh. "Security of Blind Signatures Revisited". In: *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*. 2012, pp. 662–679.

[38] The Aragon Association. *Aragon*. URL: `https://aragon.org/` (visited on 04/19/2019).

[39] B. Yu et al. "Platform-Independent Secure Blockchain-Based Voting System". In: *Information Security - 21st International Conference, ISC 2018, Guildford, UK, September 9-12, 2018, Proceedings*. 2018, pp. 369–386.