# A Treasury System for Cryptocurrencies: Enabling Better Collaborative Intelligence

Bingsheng Zhang
Lancaster University, UK
b.zhang2@lancaster.ac.uk

Roman Oliynykov
Input Output Hong Kong
roman.oliynykov@iohk.io

Hamed Balogun
Lancaster University, UK
h.balogun@lancaster.ac.uk

***Abstract*—A treasury system is a community-controlled and decentralized collaborative decision-making mechanism for sustainable funding of blockchain development and maintenance. During each treasury period, project proposals are submitted, discussed, and voted for; top-ranked projects are funded from the treasury. The Dash governance system is a real-world example of such kind of systems. In this work, we, for the first time, provide a rigorous study of the treasury system. We modelled, designed, and implemented a provably secure treasury system that is compatible with most existing blockchain infrastructures, such as Bitcoin, Ethereum, etc. More specifically, the proposed treasury system supports liquid democracy/delegative voting for better collaborative intelligence. Namely, the stake holders can either vote directly on the proposed projects or delegate their votes to experts. Its core component is a distributed universally composable secure end-to-end verifiable voting protocol. The integrity of the treasury voting decisions is guaranteed even when all the voting committee members are corrupted. To further improve efficiency, we proposed the world's first honest verifier zero-knowledge proof for unit vector encryption with logarithmic size communication. This partial result may be of independent interest to other cryptographic protocols. A pilot system is implemented in Scala over the Scorex 2.0 framework, and its benchmark results indicate that the proposed system can support tens of thousands of treasury participants with high efficiency.**

## I. Introduction

Following the success of Bitcoin, a great number of new cryptocurrencies and blockchain platforms are emerging on almost daily basis. Blockchains have become largely ubiquitous across various sectors, e.g., technology, academia, medicine, economics and finance, etc. A key feature expected from cryptocurrencies and blockchain systems is the absence of a centralized control over the operation process. That is, blockchain solutions should neither rely on "trusted parties or powerful minority" for their operations, nor introduce such (centralisation) tendencies into blockchain systems. Decentralization not only offers better security guarantees by avoiding *single point of failure*, but may also enable enhanced user privacy techniques. On the other hand, real-world blockchain systems require steady funding for continuous development and maintenance of the systems. Given that blockchain systems are decentralized systems, their maintenance and developmental funding should also be void of centralization risks. Therefore, secure and "community-inclusive" long-term sustainability of funding is critical for the health of blockchain platforms.

Network and Distributed Systems Security (NDSS) Symposium 2019
24-27 February 2019, San Diego, CA, USA
ISBN 1-891562-55-X
https://dx.doi.org/10.14722/ndss.2019.23024
www.ndss-symposium.org

In the early years, the development of cryptocurrencies, such as Bitcoin, mainly rely on patron organizations and donations. Recently, an increasing number of cryptocurrencies are funded through *initial coin offering* (ICO) – a popular crowd-funding mechanism to raise money for the corresponding startups or companies. A major drawback of donations and ICOs is that they lack sustainable funding supply. Consequently, they are not suitable as long-term funding sources for cryptocurrency development due to the difficulty of predicting the amount of funds needed (or that will be available) for future development and maintenance. Alternatively, some cryptocurrency companies, such as *Zcash Electric Coin Company*, take certain percentage of hair-cut/tax (a.k.a. founders reward) from the miners' reward. This approach would provide the companies a more sustainable funding source for long-term planning of the cryptocurrency development.

Nevertheless, the aforementioned development funding approaches have risks of centralization in terms of decision-making on the development steering. Only a few people (in the organisation or company) participate in the decision-making process on how the available funds will be used. However, the decentralized architecture of blockchain technologies makes it inappropriate to have a centralized control of the funding for secure development processes. Sometimes disagreement among the organisation members may lead to catastrophic consequences. Examples include the splitting of Ethereum and Ethereum Classic as well as Bitcoin and Bitcoin Cash.

Ideally, all cryptocurrency stake holders are entitled to participate in the decision-making process on funding allocation. This democratic type of community-inclusive decentralized decision-making enables a better *collaborative intelligence*. The concept of *treasury system* has been raised to address the highlighted issue. A treasury system is a community controlled and decentralized collaborative decision-making mechanism for sustainable funding of the underlying blockchain development and maintenance. The Dash governance system [**?**] is a real-world example of such systems. A treasury system consists of iterative treasury periods. During each treasury period, project proposals are submitted, discussed, and voted for; top-ranked projects are then funded. However, the Dash governance system has a few potential theoretical drawbacks. i) It does not offer ballot privacy to the voters (a.k.a. masternodes) [**?**]. Therefore, the soundness of any funding decision might be ill-affected. For instance, the masternodes may be subject to coercion. ii) It fails to effectively utilize the knowledge of community experts in the decision-making process. This is because the system can only support very basic type of voting schemes, and the voting power of experts are limited.

In this work, we propose to use a different approach – *liquid democracy* – to achieve better collaborative intelligence. Liquid democracy (also known as delegative democracy [**?**]) is an hybrid of direct democracy and representative democracy. It provides the benefits of both systems (whilst doing away with their drawbacks) by enabling organisations to take advantage of experts in a treasury voting process, as well as giving the stakeholders the opportunity to vote. For each project, a voter can either vote directly or delegate his/her voting power to an expert who is knowledgeable and renowned in the corresponding area.

**Collaborative decision-making.** The core component of a treasury system is a decision-making system that allows members of the community collectively reach some conclusions/decisions. During each treasury period, anyone can submit a proposal for projects to be funded. Due to shortage of available funds, only a few of them can be supported. Therefore, a collaborative decision-making mechanism is required. Note that in the literature, a few blockchain based e-voting schemes have been proposed. However, our treasury decision-making have a number of differences: (i) conventional e-voting scheme requires real-world identity authentication, while our treasury decision-making do not need to link voters to their real identities; (ii) in a conventional e-voting scheme, typically, each voter has one vote, while in our treasury decision-making, the voting power is proportional to the corresponding stake; (iii) our treasury decision-making supports liquid democracy with privacy assurance, while no other known e-voting scheme can support liquid democracy with provable security.

Proper selection of the voting scheme allows maximizing the number of voters satisfied by the voting results as well as minimizing voters' effort. In practice, there are two commonly used voting schemes: i) *preferential* or *ranked voting* and ii) *approval voting*. An extension of approval voting is the "Yes-No-Abstain" voting, where the voters express "Yes/No/Abstain" opinion for each proposal. Recent theoretical analysis of this election rule with variable number of winners, called *Fuzzy threshold voting* [**?**], shows advantages of this voting scheme for treasury application. Therefore, we will adopt this voting scheme in our treasury system. Nevertheless, we emphasize that a different voting scheme can be deployed to our treasury system without significantly changing the underlying cryptographic protocols.

**Our contributions.** In this work, we aim to resolve the funding sustainability issue for long-term cryptocurrency development and maintenance by proposing a novel treasury system. The proposed treasury system is compatible with most existing off-the-shelf cryptocurrencies/blockchain platforms, such as Bitcoin and Ethereum. We highlight the major contributions of this work as follows.

- For the first time, we provide a rigorous security modeling for a blockchain-based treasury voting system that supports liquid democracy/delegative voting. More specifically, we model the voting system in the well-known *Universally Composable* (UC) framework [**?**] via an ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k,n,m}$. The functionality interacts with a set voters and experts as well as $k$ voting committee members. It allows the voters to either delegate their voting power to some experts or vote directly on the project. If at least $t$ out of $k$ voting committee members are honest, the functionality guarantees termination. Even in the extreme case, when all the voting committee members are corrupted, the integrity of the voting result is still ensured; however, in that case we don't guarantee protocol termination.
- We propose an efficient design of the treasury system. The system collects funding via three potential sources: (i) Minting new coins; (ii) Taxation from miners' reward; (iii) Donations or charity. In an iterative process, the treasury funds accumulate over time, and the projects are funded periodically. Each treasury period consists of pre-voting epoch, voting epoch, and post-voting epoch, which can be defined in terms of number of blockchain blocks. In the pre-voting epoch, project proposals are submitted, and the voters/experts are registered. In the voting epoch, the voting committee is selected; after that, they jointly generate the voting key for the treasury period. The voters and experts then cast their ballots. In the post-voting epoch, the voting committee computes and signs the treasury decision. Winning proposals will then be funded. Any stakeholder in the community can participate in the treasury voting, and their voting power is proportional to their possessed stake. In our system, we distinguish coin ownership from stake ownership. That is, the owner of a coin can be different from the owner of the coin's stake. This allows blockchain-level stake delegation without transferring the ownership of the coin. It means that the user can delegate his/her stake to someone else without risk of losing the ultimate control of the coin(s). To achieve this, we introduced stake ownership verification mechanism using the payload of a coin. (Without loss of generality, we assume a coin has certain storage field for non-transactional data.)
- We proposed the world's first honest verifier zero-knowledge proof/argument for unit vector encryption with logarithmic size communication. Conventionally, to show a vector of ElGamal ciphertexts element-wise encrypt a unit vector, Chaum-Pedersen proofs [**?**] are used to show each of the ciphertexts encrypts either $0$ or $1$ (via Sigma OR composition) and the product of all the ciphertexts encrypts $1$. Such kind of proof is used in many well-known voting schemes, e.g., Helios. However, the proof size is linear in the length of the unit vector, and thus the communication overhead is quite significant when the unit vector length becomes larger. In this work, we propose a novel special honest verifier ZK (SHVZK) proof/argument for unit vector that allows the prover to convince the verifier that a vector of ciphertexts $(C_0, \ldots, C_{n-1})$ encrypts a unit vector $\mathbf{e}_i^{(n)}$, $i \in [0, n-1]$ with $O(\log n)$ proof size. The proposed SHVZK protocol can also be Fiat-

Shamir transformed to a non-interactive ZK (NIZK) proof in the random oracle model.

- We provide prototype implementation [?] of the proposed treasury system for running and benchmarking in the real world environment. Our implementation is written in Scala programming language over Scorex 2.0 framework and uses TwinsChain consensus for keeping the underlying blockchain. Main functionality includes proposal submission, registration of voters, experts, voting committee members and their corresponding deposit lock, randomized selection of the voting committee members among voters, distributed key generation (6-round protocol), ballots casting, joint decryption with recovery in case of faulty committee members (4-round protocol), randomness generation for the next treasury period (3-round protocol), reward payments, deposit paybacks, and penalties for faulty actors. All implemented protocols are fully decentralized and resilient up to 50% of malicious participants. During verification we launched a testnet that consisted of 12 full nodes successfully operating tens of treasury periods with different parameters.

## II. Preliminaries

**Notations.** Throughout this paper, we will use the following notations. Let $\lambda \in \mathbb{N}$ be the security parameter. Denote the set $\{a, a+1, \ldots, b\}$ by $[a, b]$, and let $[b]$ denote $[1, b]$. We abbreviate *probabilistic polynomial time* as PPT. By $\mathbf{a}^{(\ell)}$, we denote a length-$\ell$ vector $(a_1, \ldots, a_\ell)$. When $S$ is a set, $s \leftarrow S$ stands for sampling $s$ uniformly at random from $S$. When $A$ is a randomised algorithm, $y \leftarrow A(x)$ stands for running $A$ on input $x$ with a fresh random coin $r$. When needed, we denote $y := A(x; r)$ as running $A$ on input $x$ with the explicit random coin $r$. Let $\text{poly}(\cdot)$ and $\mathsf{negl}(\cdot)$ be a polynomially-bounded function and negligible function, respectively.

**The blockchain abstraction.** Without loss of generality, we abstract the underlying blockchain platform encompasses the following concepts.

∘ *Coin.* We assume the underlying blockchain platform has the notion of *Coins* or its equivalent. Each coin can be spent only once, and all the value of coin must be consumed. As depicted in Fig. **??**, each coin consists of the following 4 attributes:

- **Coin ID**: It is an implicit attribute, and every coin has a unique ID that can be used to identify the coin.
- **Value**: It contains the value of the coin.
- **Cond**: It contains the conditions under which the coin can be spent.
- **Payload**: It is used to store any non-transactional data.

∘ *Address.* We also generalize the concept of *address*. Conventionally, an address is merely a public key, $\mathsf{pk}$, or hash of a public key, $h(\mathsf{pk})$. To create coins associated with the address, the spending condition of the coin should be defined as a valid signature under the corresponding public key $\mathsf{pk}$ of the address. In this work, we define an address as a generic representation of some spending condition. Using the recipient's address, a sender is able to create a new coin whose spending condition is the one that the recipient intended; therefore, the recipient may spend the coin later.

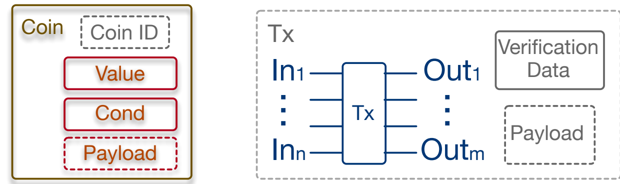


Fig. 1: Coin and transaction structure.

∘ *Transaction.* Each transaction takes one or more (unspent) coins, denoted as $\{\mathsf{In}_i\}_{i\in[n]}$, as input, and it outputs one or more (new) coins, denoted as $\{\mathsf{Out}_j\}_{j\in[m]}$. Except special transactions, the following condition holds:

$$\sum_{i=1}^{n} \mathsf{In}_i.\mathsf{Value} \geq \sum_{j=1}^{m} \mathsf{Out}_j.\mathsf{Value}$$

and the difference is interpreted as transaction fee. As shown in Fig. **??**, the transaction has a Verification Data field that contains necessary verification data to satisfy all the spending conditions of the input coins $\{\mathsf{In}_i\}_{i\in[n]}$. In addition, each transaction also has a Payload field that can be used to store any non-transactional data. We denote a transaction as $\mathsf{Tx}(A; B; C)$, where $A$ is the set of input coins, $B$ is the set of output coins, and $C$ is the Payload field. Note that the verification data is not explicitly described for simplicity.

**Universal composability.** We model our system security under the standard *Universal Composability* (UC) framework. The protocol is represented as interactive Turing machines (ITMs), each of which represents the program to be run by a participant. Adversarial entities are also modeled as ITMs.

We distinguish between ITMs (which represent static objects, or programs) and *instances of ITMs* (*ITIs*), that represent interacting processes in a running system. Specifically, an ITI is an ITM along with an identifier that distinguishes it from other ITIs in the same system. The identifier consists of two parts: A session-identifier (SID) which identifies which protocol instance the ITI belongs to, and a party identifier (PID) that distinguishes among the parties in a protocol instance. Typically, the PID is also used to associate ITIs with "parties" that represent some administrative domains or physical computers.

The model of computation consists of a number of ITIs that can write on each other's tapes in certain ways (specified in the model). The pair (SID,PID) is a unique identifier of the ITI in the system. With one exception (discussed later), we assume that all ITMs are PPT.

We consider the security of the voting system in the UC framework with static corruption in the random oracle (RO) model. The security is based on the indistinguishability between real/hybrid world executions and ideal world executions, i.e., for any possible PPT real/hybrid world adversary $\mathcal{A}$ we will construct an ideal world PPT simulator $\mathcal{S}$ that can present an indistinguishable view to the environment $\mathcal{Z}$ operating the protocol.

**Additively homomorphic encryption.** In this work, we adopt the well known threshold lifted ElGamal encryption scheme as the candidate of the threshold additively homomorphic public key cryptosystem. Let $\mathsf{Gen}^{\mathsf{gp}}(1^\lambda)$ be the group generator that takes as input the security parameter $\lambda \in \mathbb{N}$, and outputs the group parameters $\mathsf{param}$, which define a multiplicative cyclic group $\mathbb{G}$ with prime order $p$, where $|p| = \lambda$. We assume the DDH assumption holds with respect to the group generator $\mathsf{Gen}^{\mathsf{gp}}$. More specifically, the additively homomorphic cryptosystem $\mathsf{HE}$ consists of algorithms $(\mathsf{KeyGen}^{\mathsf{E}}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Dec})$ as follows:

- $\mathsf{KeyGen}^{\mathsf{E}}(\mathsf{param})$: pick $\mathsf{sk} \leftarrow \mathbb{Z}_q^*$ and set $\mathsf{pk} := h = g^{sk}$, and output $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}_{\mathsf{pk}}(m; r)$: output $e := (e_1, e_2) = (g^r, g^m h^r)$.
- $\mathsf{Add}(c_1, \ldots, c_\ell)$: output $c := (\prod_{i=1}^{\ell} c_{i,1}, \prod_{i=1}^{\ell} c_{i,2})$.
- $\mathsf{Dec}_{\mathsf{sk}}(e)$: output $\mathsf{Dlog}(e_2 \cdot e_1^{-\mathsf{sk}})$, where $\mathsf{Dlog}(x)$ is the discrete logarithm of $x$. (Note that since $\mathsf{Dlog}(\cdot)$ is not efficient, the message space should be a small set.)

Lifted ElGamal encryption is additively homomorphic, i.e.

$$\mathsf{Enc}_{\mathsf{pk}}(m_1; r_1) \cdot \mathsf{Enc}_{\mathsf{pk}}(m_2; r_2) = \mathsf{Enc}_{\mathsf{pk}}(m_1 + m_2; r_1 + r_2) \ .$$

## III. The Treasury System

**Entities.** As mentioned before, the core of a treasury system is a collaborative decision-making process, and all the stake holders are eligible to participate. Let $k, \ell, n, m$ be integers in $\mathrm{poly}(\lambda)$. The stake holders may have one or more of the following roles.

- The project owners $\mathcal{O} := \{\mathsf{O}_1, \ldots, \mathsf{O}_k\}$ are a set of stake holders that have proposed project for support.
- The voting committees $\mathcal{C} := \{\mathsf{C}_1, \ldots, \mathsf{C}_\ell\}$ are a set of stake holders that are responsible for generating the voting public key and announcing the voting result.
- The voters $\mathcal{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$ are a set of stake holders that lock certain amount of stake to participate.
- The experts $\mathcal{E} := \{\mathsf{E}_1, \ldots, \mathsf{E}_m\}$ are a special type of voters that have specialist knowledge and expertise in some field.

**Enabling stake delegation.** In our treasury system, the voting power of a voter is proportional to the corresponding locked stake value. We distinguish between the ownership of a stake and the ownership of the actual coin; namely, the stake of a coin can be "owned" by a user other than the coin owner. This feature allows us to delegate the stake of a coin to someone else without transferring the ownership of the coin. To achieve this, we introduce a stake attribute, denoted as S-Attr, that can be attached to the Payload of a coin. The user who can provide the required data that satisfies the condition(s) in the S-Attr is able to claim the stake of the coin. Of course, the stake of an unspent coin can only be claimed at most once at any moment. In practice, to ensure this, additional checks should be executed. If the user A wants to delegate the stake of a coin to the user B, he simply needs to put the user B's desired S-Attr in the Payload of the coin. Note that this type of delegation is persistent in the sense that if the coin is not consumed, the S-Attr of the coin remains the same. This feature allows users to stay offline while the stake of their coins can still be used in the treasury process by the delegatees. However, this type of delegation only guarantees pseudonymity-based privacy level, as anyone can learn "who owns" the stake of the coin by checking the S-Attr of the coin.

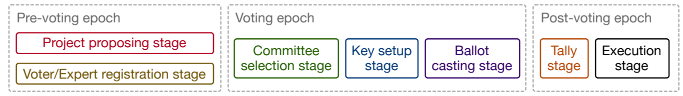


Fig. 2: Treasury system epochs.

**System overview.** A treasury system consists of iterative treasury periods. A treasury period can be divided into three epochs: pre-voting epoch, voting epoch, and post-voting epoch. As shown in Figure **??**, the pre-voting epoch includes two concurrent stages: project proposing stage and voter/expert registration stage. In the project proposing stage, the users can submit project proposals, asking for treasury funds. Meanwhile, the interested stake holders can register themselves as either voters and/or experts to participate in the decision making process by locking certain amount of their stake in the underlying cryptocurrency. The voter's voting power is proportional to his locked stake; while, the expert's voting power is proportional to the amount of voting power delegated to him. (We will explain delegation in details later.) Analogously, the voter's (resp. expert's) treasury reward is proportional to his locked stake (resp. his received delegations).

At the beginning of the voting epoch, there is a voting committee selection stage, during which, a set of voting committee members will be randomly selected from the registered voters who are willing to be considered for selection to the committee. The probability of being selected is proportional to locked stake. After the voting committee members are selected, they jointly run a distributed key generation protocol to setup the election public key. The voters and experts can then submit their ballots in the ballot casting stage. Note that the voters can either delegate their voting powers to some expert or vote directly on the projects. For each project, voters can delegate to different experts. At the post-voting epoch, the voting committee members jointly calculate and announce the tally result on the blockchain. Finally, in the execution stage, the winning projects are funded, and the voters, experts and voting committee members are rewarded (or punished) accordingly. These transactions will be jointly signed and executed by the voting committee. Meanwhile, the committee members also jointly commit to a random seed, which will be used to select a new voting committee in the next treasury period.

**Treasury funding sources.** As earlier motivated, treasury funding, perhaps is the most crucial ingredient in a decentralised community-controlled decision-making system. It must not only be regular, but also sourced from decentralised means. That is, source of funding for treasury system should not introduce centralisation into the system. To this end, desirable properties from the funding sources are secure, sustainable and decentralized.

We note that although not all potential funding sources possess these properties, a clever combination of some of these sources satisfy the set out requirement. Therefore, we propose 3 major sources of funding for the treasury system.

- *Taxation/Haircut from block reward:* Most blockchain platforms offer block rewards (including transaction fees) to proposers of new blocks, incentivizing honest behaviour. A fraction of such block rewards can be taken and contributed to the decentralised treasury. This type of funding source is sustainable as long as the block rewards of the underlying blockchain platform remain. However, block rewards may fluctuate over time, and it could cause unpredictability of the available funds.
- *Minting new coins:* Coin minting represents, perhaps, the most sustainable funding source of the potential sources. At the beginning of each treasury period, certain amount of new coins are created to fund projects. However, minting may cause inflation in terms of the fiat market value of the underlying cryptocurrency.
- *Donations or charity:* Donation is an opportunistic ad-hoc but unsustainable funding source. Therefore, meticulous blockchain development planning is difficult if donations is the only means of treasury funding.

**Project proposal.** To ensure input independency and eliminate unfair advantage caused by late submission, we adopt a two-stage project proposal scheme. In the first stage, the project owners $\mathsf{O}_1, \ldots, \mathsf{O}_k$ post an encryption of their project proposals (encrypted under the election public key of the previous treasury period) to the blockchain. At the end of pre-voting epoch and the beginning of the voting epoch, the voting committee of previous treasury period will jointly decrypt those project proposals (together with revealing the seed, which will be explained later).

To commit a project, the project owner needs to submit a special transaction in form of

$$\mathsf{Tx}\Big(\{\mathsf{In}_i\}_{i=1}^n; \mathsf{TCoin}; \{\textsc{Project}, \mathsf{TID}, \mathsf{P\text{-}Enc}, \mathsf{Addr}\}\Big) ,$$

where $\{\mathsf{In}_i\}_{i=1}^n$ are the input coins, and $\mathsf{TCoin}$ is a special output coin whose spending condition is defined as, the coin can only be spent according to the corresponding treasury decision (cf. Subsection "supplying the treasury", below). Moreover, the coin value $\mathsf{TCoin.Value} \geq \alpha_{\min}$, where $\alpha_{\min}$ is the minimum required fee for a project proposal to prevent *denial-of-service* attacks. In the Payload field, PROJECT is a tag that indicates it is a special project proposal transaction; $\mathsf{TID}$ is the treasury ID that is used to uniquely identify a treasury period; $\mathsf{P\text{-}Enc}$ is the encrypted project proposal, and $\mathsf{Addr}$ is the return address for the project owner to receive money if the project succeeds in getting funded.

**Voter/Expert registration.** In order to register to be a voter, a stake holder (or a set of stake holders) need(s) to submit a special *voter registration transaction* in form of

$$\mathsf{Tx}\Big(\{\mathsf{In}_i\}_{i=1}^n; \mathsf{TCoin}; \Big\{\textsc{Voter-Reg}, \mathsf{TID}, \{\mathsf{S}_i\}_{i=1}^{\ell}, \mathsf{S\text{-}Cond}, \mathsf{vk}, \mathsf{Addr}\Big\}\Big) ,$$

where $\{\mathsf{In}_i\}_{i=1}^n$ are the input coins, and $\mathsf{TCoin}$ is a special output coin whose spending condition is defined in Subsection "supplying the treasury", below. In the Payload field, VOTER-REG is a tag that indicates it is a special voter registration transaction; $\mathsf{TID}$ is the treasury ID that is used to uniquely identify a treasury period; $\{\mathsf{S}_i\}_{i=1}^{\ell}$ are the *freezed* unspent coins that will be used to claim stake value, $\mathsf{S\text{-}Cond}$ is the required data that satisfies all the stake attributes of $\{\mathsf{S}_i\}_{i=1}^{\ell}$, $\mathsf{vk}$ is a freshly generated signature key; and $\mathsf{Addr}$ is the return address for the voter to receive treasury reward. The voter's ID is defined as the hash of $\mathsf{vk}$, denoted as $\mathsf{V}_i := \mathsf{hash}(\mathsf{vk})$.

Let $\beta_{\min}$ be a predefined system parameter. To register as an expert, a stake holder (or a set of stake holders) need(s) to deposit *exact* $\beta_{\min}$ amount of coins, by submitting a special *expert registration transaction*:

$$\mathsf{Tx}\Big(\{\mathsf{In}_i\}_{i=1}^n; \mathsf{TCoin}; \{\textsc{Expert-Reg}, \mathsf{TID}, \mathsf{vk}, \mathsf{Addr}\}\Big) ,$$

where $\{\mathsf{In}_i\}_{i=1}^n$ are the input coins, and $\mathsf{TCoin}$ is a special output coin whose spending condition is defined in Subsection "supplying the treasury". Moreover, the coin value $\mathsf{TCoin.Value} \geq \beta_{\min}$. In the Payload field, EXPERT-REG is a tag that indicates it is a special expert registration transaction; $\mathsf{TID}$ is the treasury ID that is used to uniquely identify a treasury period; $\mathsf{vk}$ is a freshly generated signature key; and $\mathsf{Addr}$ is the return address for the expert to receive treasury reward.

The expert's ID is defined as the hash of $\mathsf{vk}$, denoted as $\mathsf{E}_j := \mathsf{hash}(\mathsf{vk})$. Note that the expert does not gain reward based on the amount of deposited coins, so it is not rational to deposit significantly more than $\beta_{\min}$ coins in practice.

**Voting committee selection.** At the beginning of the voting epoch, the voting committee of the previous treasury epoch jointly reveal the committed seed, seed.

Let $\mathsf{st}_i = \sum_{j=1}^{\ell} S_j.\mathsf{Value}$ for all the stake coins $S_j$ claimed in the payload of the voter registration transaction of $\mathsf{vk}_i$, i.e. $\mathsf{st}_i$ is the total stake amount claimed by $\mathsf{vk}_i$. Once seed is announced, any registered voter, who have an address $\mathsf{vk}_i$ with claimed stake $\mathsf{st}_i$, can volunteer to participate in the voting committee if the following inequality holds:

$$\mathsf{hash}\big(\mathsf{vk}_i, \mathsf{sign}_{\mathsf{sk}'_i}(\mathsf{seed})\big) \leq \mathsf{st}_i \cdot T$$

where $\mathsf{sk}'_i$ is the corresponding signing key for $\mathsf{vk}_i$, and $T$ is a pre-defined threshold. When the inequation holds, he/she can submit a special *registration transaction* in form of

$$\mathsf{Tx}\Big(\{\mathsf{In}_i\}_{i=1}^n; \mathsf{TCoin}; \Big\{\textsc{VC-Reg}, \mathsf{TID}, \overline{\mathsf{vk}}, \tilde{\mathsf{pk}}, \mathsf{sign}_{\mathsf{sk}'_i}(\mathsf{seed}), \mathsf{Addr}\Big\}\Big) ,$$

where $\{\mathsf{In}_i\}_{i=1}^n$ are the input coins, and $\mathsf{TCoin}$ is a special output coin whose spending condition is defined in Subsection "supplying the treasury", below. Moreover, the coin value $\mathsf{TCoin.Value} \geq \gamma_{\min}$. In the Payload field, VC-REG is a tag that indicates it is a special voting committee registration transaction; $\mathsf{TID}$ is the treasury ID that is used to uniquely identify a treasury period; $\overline{\mathsf{vk}}$ is a freshly generated signature verification key; $\tilde{\mathsf{pk}}$ is a freshly generated public key for a pre-defined public key cryptosystem; $\mathsf{sign}_{\mathsf{sk}'_i}(\mathsf{seed})$ is the signature of seed under the signing key corresponding to $\mathsf{vk}_i$; and $\mathsf{Addr}$ is the return address for the committee member to receive treasury reward. The threshold $T$ is properly defined to ensure that approximately $\lambda' = \omega(\log \lambda)$ (e.g., $\lambda' = \mathrm{polylog}(\lambda)$) committee members are selected, assuming constant fraction of them will be active. Note that, analogous to most *proof-of-stake* systems, $T$ needs to be updated frequently. See [?] for a common threshold/difficulty $T$ adjustment approach.

*Remark.* Jumping ahead, we will need honest majority of the voting committee to guarantee voter privacy and protocol termination. Assume the majority of the stake of all the registered voters is honest; therefore, the probability that a selected committee member is honest is $p = 1/2 + \varepsilon$ for any $\varepsilon \in (0, 1/2]$. Let $X$ be the number of malicious committee members selected among all $\lambda'$ committee members. Since $\lambda' = \omega(\log \lambda)$, by Chernoff bound, for $\delta = 2\varepsilon/(1-2\varepsilon)$:

$$\begin{aligned}\Pr[X \geq \lambda'/2] &= \Pr[X \geq (1+\delta)(1/2-\varepsilon)\lambda'] \\ &< \exp(-\delta^2(1/2-\varepsilon)\lambda'/4) \\ &= \frac{1}{\exp(\omega(\log \lambda))} = \mathsf{negl}(\lambda)\end{aligned}$$

**Supplying the treasury.** Treasury funds are accumulated via a collection of coins. For example, the taxation/haircut of the block reward can be collected through a special transaction at the beginning of each block. The output of this type of transactions are new coins, whose spending condition, $\mathsf{Cond}$, specifies that the coin can only be spent according to the corresponding treasury decision. As will be explained later in details, the treasury funds will be distributed in forms of transactions jointly made by the corresponding voting committee; therefore, the coins dedicated to certain treasury period must allow the voting committee in that treasury period to jointly spend. More specifically, there are $\lambda'$ committee members selected at the beginning of the voting epoch of each treasury period. Let $\mathsf{seed}_{\mathsf{TID}_i}$ denote the seed opened in the treasury period indexed by $\mathsf{TID}_i$. Let $\{\overline{\mathsf{vk}}_j\}_{j=1}^{\ell}$ be the set of signature verification keys in the valid committee registration transactions proposed by $\mathsf{vk}_i$ such that the condition $\mathsf{hash}\big(\mathsf{vk}_i, \mathsf{sign}_{\mathsf{sk}'_i}(\mathsf{seed})\big) \leq \mathsf{st}_i \cdot T$ holds. The treasury coin can be spent in a transaction if majority of the signatures w.r.t. $\{\overline{\mathsf{vk}}_j\}_{j=1}^{\ell}$ are present.

**Handling the treasury specific data in the payload.** Note that typically the underlying blockchain transaction validation rules do not take into account of the content stored in the payload of a transaction. Therefore, additional checks are needed for the treasury specific transactions. More specifically, we verify the payload data of those transactions with additional algorithms. In particular, a coin must be *frozen* during the entire treasury period in order to claim its stake. This can be done by, for example, adding extra constrain in spending condition, saying that the coin cannot be spent until the certain block height, which is no earlier than the end of the treasury period. Furthermore, the stake of one coin can only be claimed once during each treasury period.

**Decision making.** During the decision making, the voting committee members, the voters, and the experts follow the protocol description in Sec. **??**, below. It covers the key generation stage, the ballot casting stage, and the tally stage. In terms of security, as shown before, with overwhelming probability, the majority of the committee members are honest, which can guarantee voter privacy and protocol termination. In an unlikely extreme case, where all the voting committee members are corrupted, our voting scheme can still ensure the integrity of the voting result. If a cheating voting committee member is detected, she will lose all her deposit.

For each project, the voters/experts need to submit an independent ballot. The voter can either delegate his voting power to some expert or directly express his opinion on the project; whereas, the expert shall only vote directly on the project. In our prototype, we adopt the "YES-NO-ABSTAIN" type of voting scheme. More specifically, after the voting, the project proposals are scored based on the number of yes votes minus the number of no votes. Proposals that got at least 10% (of all votes) of the positive difference are shortlisted, and all the remaining project proposals are discarded. Shortlisted proposals are ranked according to their score, and the top ranked proposals are funded in turns until the treasury fund is exhausted. Each of the voting committee members will then sign the treasury decision and treasury transactions, and those transactions are valid if it is signed by more than $t$-out-of-$k$ voting committee members.

**Post-voting execution.** Certain proportion (e.g. 20%) of the treasury fund will be used to reward the voting committee members, voters and experts. The voting committee members $\mathsf{C}_\ell \in \mathcal{C}$ will receive a fix amount of reward, denoted as $\zeta_1$. Note that as the voting committee members are required to perform more actions in the next treasury period, their reward will only be transferred after the completion of those actions at the end of pre-voting epoch in the next treasury period. The voter $\mathsf{V}_i \in \mathcal{V}$ will receive reward that is proportional to his/her deposited amount, denoted as $\zeta_2 \cdot \mathsf{st}_i$, where $\mathsf{st}_i$ is the amount of the stake claimed by $\mathsf{V}_i$. The expert $\mathsf{E}_j \in \mathcal{E}$ will receive reward that is proportional to his/her received delegations, denoted as $\zeta_3 \cdot D_j$, where $D_j$ is the amount of delegations that $\mathsf{E}_j$ has received. Meanwhile, if a voting committee member cheats or an expert fails to submit a valid ballot, he/she will lose the deposited coin as a punishment. In addition, the voting committee members will jointly generate and commit to a random seed for the next treasury period, in a protocol depicted as follows. To generate and commit a random seed, voting committee members $\mathsf{C}_\ell$, $\ell \in [k]$ needs to invoke a coin flipping protocol. However, the cost of such a protocol is very small when they already jointly setup a public key $\mathsf{pk}$. More specifically, each voting committee members $\mathsf{C}_\ell$, $\ell \in [k]$ will pick a random group element $R_\ell \leftarrow \mathbb{G}$ and post the encryption of it, $C_\ell \leftarrow \mathsf{Enc}_{\mathsf{pk}}(R_\ell)$ to the blockchain. $C := \prod_{\ell=1}^{k} C_\ell$ is defined as the committed/encrypted seed for the next treasury period. Note that $C$ can be jointly decrypted as far as majority of the voting committee members are honest, and the malicious voting committee members cannot influence the distribution of the seed.

**Partitionary budgeting.** The main goal of treasury is decentralized community-driven self-sustainable cryptocurrency development through projects funding and adoption. The naive approach is to select projects for funding by ranking all submitted proposals according to the number of votes they get and take a number of projects whose total budget does not exceed the treasury budget. However, there exists a risk of underfunding vital areas due to numerous project submissions and inflated discussions on some other areas. We can categorize proposals and allocate a certain amount of treasury funding for each category to independently guarantee funds to every vital area.

Analysis of existing blockchain development funding [**?**] reveal marketing, PR, integration, software development and

The ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k,n,m}$

The functionality $\mathcal{F}_{\text{VOTE}}^{t,k,n,m}$ interacts with a set of voting committees $\mathcal{C} := \{\mathsf{C}_1, \ldots, \mathsf{C}_k\}$, a set of voters $\mathcal{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$, a set of experts $\mathcal{E} := \{\mathsf{E}_1, \ldots, \mathsf{E}_m\}$, and the adversary $\mathcal{S}$. It is parameterized by a delegation calculation algorithm $\mathsf{DelCal}$ (described in Fig. **??**) and a tally algorithm $\mathsf{TallyAlg}$ (described in Fig. **??**) and variables $\phi_1, \phi_2$, $\tau$, $J_1, J_2, J_3$, $T_1$ and $T_2$. Denote $\mathcal{C}_{\mathsf{cor}}$ and $\mathcal{C}_{\mathsf{honest}}$ as the set of corrupted and honest voting committees, respectively.
Initially, $\phi_1 = \emptyset$, $\phi_2 = \emptyset$, $\tau = \emptyset$, $J_1 = \emptyset$, $J_2 = \emptyset$, and $J_3 = \emptyset$.

**Preparation:**

- Upon receiving $(\textsc{Init}, \mathsf{sid})$ from the voting committee $\mathsf{C}_i \in \mathcal{C}$, set $J_1 := J_1 \cup \{\mathsf{C}_i\}$, and send a notification message $(\textsc{InitNotify}, \mathsf{sid}, \mathsf{C}_i)$ to the adversary $\mathcal{S}$.

**Voting/Delegation:**

- Upon receiving $(\textsc{Vote}, \mathsf{sid}, v_i)$ from the expert $\mathsf{E}_i \in \mathcal{E}$, if $|J_1| < t$, ignore the request. Otherwise, record $(\mathsf{E}_i, \textsc{Vote}, v_i)$ in $\phi_1$; send a notification message $(\textsc{VoteNotify}, \mathsf{sid}, \mathsf{E}_i)$ to the adversary $\mathcal{S}$. If $|\mathcal{C}_{\mathsf{cor}}| \geq t$, then additionally send a message $(\textsc{Leak}, \mathsf{sid}, \mathsf{E}_i, \textsc{Vote}, v_i)$ to the adversary $\mathcal{S}$.
- Upon receiving $(\textsc{Cast}, \mathsf{sid}, v_j, \alpha_j)$ from the voter $\mathsf{V}_j \in \mathcal{V}$, if $|J_1| < t$, ignore the request. Otherwise, record $(\mathsf{V}_j, \textsc{Cast}, v_j, \alpha_j)$ in $\phi_2$; send a notification message $(\textsc{CastNotify}, \mathsf{sid}, \mathsf{V}_j, \alpha_j)$ to the adversary $\mathcal{S}$. If $|\mathcal{C}_{\mathsf{cor}}| \geq t$, then additionally send a message $(\textsc{Leak}, \mathsf{sid}, \mathsf{V}_j, \textsc{Cast}, v_j)$ to the adversary $\mathcal{S}$.

**Tally:**

- Upon receiving $(\textsc{DelCal}, \mathsf{sid})$ from the voting committee $\mathsf{C}_i \in \mathcal{C}$, set $J_2 := J_2 \cup \{\mathsf{C}_i\}$, and send a notification message $(\textsc{DelCalNotify}, \mathsf{sid}, \mathsf{C}_i)$ to the adversary $\mathcal{S}$.
- If $|J_2 \cup \mathcal{C}_{\mathsf{honest}}| + |\mathcal{C}_{\mathsf{cor}}| \geq t$, send $(\textsc{LeakDel}, \mathsf{sid}, \mathsf{DelCal}(\mathcal{E}, \phi_2))$ to $\mathcal{S}$.
- If $|J_2| \geq t$, set $\delta \leftarrow \mathsf{DelCal}(\mathcal{E}, \phi_2)$.
- Upon receiving $(\textsc{Tally}, \mathsf{sid})$ from the voting committee $\mathsf{C}_i \in \mathcal{C}$, set $J_3 := J_3 \cup \{\mathsf{C}_i\}$, and send a notification message $(\textsc{TallyNotify}, \mathsf{sid}, \mathsf{C}_i)$ to the adversary $\mathcal{S}$.
- If $|J_3 \cup \mathcal{C}_{\mathsf{honest}}| + |\mathcal{C}_{\mathsf{cor}}| \geq t$, send $(\textsc{LeakTally}, \mathsf{sid}, \mathsf{TallyAlg}(\mathcal{V}, \mathcal{E}, \phi_1, \phi_2, \delta))$ to $\mathcal{S}$.
- If $|J_3| \geq t$, set $\tau \leftarrow \mathsf{TallyAlg}(\mathcal{V}, \mathcal{E}, \phi_1, \phi_2, \delta)$.
- Upon receiving $(\textsc{ReadTally}, \mathsf{sid})$ from any party, if $\delta = \emptyset \wedge \tau = \emptyset$ ignore the request. Otherwise, return $(\textsc{ReadTallyReturn}, \mathsf{sid}, (\delta, \tau))$ to the requester.

Fig. 3: The ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k,n,m}$

organisational costs are most prominent categories. Considering this and general business development rules, we propose to include (at least) the following categories.

- **Marketing.** This covers activities devoted to cryptocurrency market share growth; market analysis, advertisement, conferences, etc. The vastness of the area demands this category should take the biggest percent of the funding budget.
- **Technology adoption.** This includes costs needed for wider spreading of cryptocurrency; integration with various platforms, websites and applications, deployment of ATMs etc.
- **Development and security.** This includes costs allocated for funding core and non-core development, security incident response, patch management, running testnets, as well as similar critical technology areas.
- **Support.** This category includes user support, documentation, maintaining of web-infrastructure needed for the community and other similar areas.
- **Organization and management.** This category includes costs on team coordination and management, legal support, etc.
- **General.** This includes projects not covered by the earlier categories, e.g., research on prospective technologies for cryptocurrency application, external security audit, collaboration with other communities, charity and so on.

It should be noted that the given list of categories is not final, and treasury deployment in cryptocurrencies will take into account specific of a given solution based on its development effort.

Nevertheless, having such an approach guarantees that critical areas for cryptocurrency routine operation, support and development will always get funding via treasury, which in turn, guarantees cryptocurrency self-sustainability.

## IV. THE PROPOSED VOTING SCHEME

### A. *Security modeling*

The entities involved in the voting schemes are a set of voting committee members $\mathcal{C} := \{\mathsf{C}_1, \ldots, \mathsf{C}_k\}$, a set of voters $\mathcal{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$, and a set of experts $\mathcal{E} := \{\mathsf{E}_1, \ldots, \mathsf{E}_m\}$. We consider the security of our treasury voting scheme in the UC framework with static corruption. The security is based on the indistinguishability between real/hybrid world executions and ideal world executions, i.e., for any PPT real/hybrid world adversary $\mathcal{A}$ we will construct an ideal world PPT simulator $\mathcal{S}$ that can present an indistinguishable view to the environment $\mathcal{Z}$ operating the protocol.

**The Ideal world execution.** In the ideal world, the voting committee $\mathcal{C}$, the voters $\mathcal{V}$, and the experts $\mathcal{E}$ only communicate to an ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k,m,n}$ during the execution. The ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k,m,n}$ accepts a number of commands from $\mathcal{C}, \mathcal{V}, \mathcal{E}$. At the same time it informs the adversary of certain actions that take place and also is influenced by the adversary to elicit certain actions. The ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k,m,n}$ is depicted in Fig. **??**, and it consists of three phases: Preparation, Voting/Delegation, and Tally.

*Preparation phase.* During the preparation phase, the voting committees $\mathsf{C}_i \in \mathcal{C}$ need to initiate the voting process by sending $(\textsc{Init}, sid)$ to the ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k,m,n}$. The voting will not start until all the committees have participated the preparation phase.

*Voting/Delegation phase.* During the voting/delegation phase, the expert $\mathsf{E}_i \in \mathcal{E}$ can vote for his choice $v_i$ by sending $(\textsc{Vote}, sid, v_i)$ to the ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k,m,n}$. Note that the voting choice $v_i$ is leaked only when majority of the voting committees are corrupted. The voter $\mathsf{V}_j \in \mathcal{V}$, who owns $\alpha_j$ stake, can either vote directly for his choice $v_j$ or delegate his voting power to an expert $\mathsf{E}_i \in \mathcal{E}$. Similarly, when all the voting committees are corrupted, $\mathcal{F}_{\text{VOTE}}^{t,k,m,n}$ leaks the voters' ballots to the adversary $\mathcal{S}$.

*Tally phase.* During tally phase, the voting committee $\mathsf{C}_i \in \mathcal{C}$ sends $(\textsc{DelCal}, \mathsf{sid})$ to the ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k,m,n}$ to calculate and reveal the delegations received by each expert. After that, they then send $(\textsc{Tally}, sid)$ to the ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k,m,n}$ to open the tally. Once all the committees have opened the tally, any party can read the tally by sending $(\textsc{ReadTally}, sid)$ to $\mathcal{F}_{\text{VOTE}}^{t,k,m,n}$. Note that due to the nature of threshold cryptography, the adversary $\mathcal{S}$ can see the voting tally result before all the honest parties. Hence, the adversary can refuse to open the tally depending on the tally result. The tally algorithm TallyAlg is described in Fig. **??**.

**The real/hybrid world execution.** In the real/hybrid world, the treasury voting scheme utilises a number of supporting components. Those supporting components are modelled as ideal functionalities. First of all, we need a blockchain functionality $\mathcal{F}_{\text{LEDGER}}$ [**?**] to model the underlying blockchain infrastructure that the treasury system is built on. We then use the key generation functionality $\mathcal{F}_{\text{DKG}}^{t,k}$ [**?**] for threshold key generation of the underlying public key crypto system. Finally, a global clock functionality $\mathcal{G}_{\text{CLOCK}}$ [**?**] is adopted to model the synchronised network environment. Let $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ denote the output of the environment $\mathcal{Z}$ when interacting with parties running the protocol $\Pi$ and real-world adversary $\mathcal{A}$. Let $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote output of $\mathcal{Z}$ when running protocol $\phi$ interacting with the ideal functionality $\mathcal{F}$ and the ideal adversary $\mathcal{S}$.

*Definition 1:* We say that a protocol $\Pi$ UC-realizes $\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security we have $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

### B. The voting scheme

Let $m$ be the number of experts and $n$ be the number of voters. Let $\mathbf{e}_i^{(m)} \in \{0,1\}^m$ be the unit vector where its $i$-th coordinate is 1 and the rest coordinates are 0. We also abuse the notation to denote $\mathbf{e}_0^{(\ell)}$ as an $\ell$-vector contains all 0's. We use $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{e}_i^{(\ell)})$ to denote coordinate-wise encryption of $\mathbf{e}_i^{(\ell)}$, i.e. $\mathsf{Enc}_{\mathsf{pk}}(e_{i,1}^{(\ell)}), \ldots, \mathsf{Enc}_{\mathsf{pk}}(e_{i,\ell}^{(1)})$, where $\mathbf{e}_i^{(\ell)} = (e_{i,1}^{(\ell)}, \ldots, e_{i,\ell}^{(\ell)})$.

*1) Vote encoding:* In our scheme, we encode the vote into a (unit) vector. Let $\mathsf{encode}^{\mathsf{E}}$ and $\mathsf{encode}^{\mathsf{V}}$ be the vote encoding algorithm for the expert and voter, respectively. For an expert, upon receiving input $x \in \{\textsc{Yes}, \textsc{No}, \textsc{Abstain}\}$, the $\mathsf{encode}^{\mathsf{E}}$ returns 100, 010, 001 for $\textsc{Yes}, \textsc{No}, \textsc{Abstain}$, respectively. For a voter, the input is $y \in \{\mathsf{E}_1, \ldots, \mathsf{E}_m\} \cup \{\textsc{Yes}, \textsc{No}, \textsc{Abstain}\}$. When $y = \mathsf{E}_i$, $i \in [m]$, it means that the voter delegate his/her voting power to the expert $\mathsf{E}_i$. When $y \in \{\textsc{Yes}, \textsc{No}, \textsc{Abstain}\}$, it means that the voter directly vote on the project. The $\mathsf{encode}^{\mathsf{V}}$ returns a unit vector of length $(m+3)$, denoted as $v$, such that $v = e_i^{(m+3)}$ if $y = \mathsf{E}_i$, for $i \in [m]$; and $v$ is set to $e_{m+1}^{(m+3)}$, $e_{m+2}^{(m+3)}$, and $e_{m+3}^{(m+3)}$ if $y$ is $\textsc{Yes}, \textsc{No}, \textsc{Abstain}$, respectively.

**Algorithm DelCal**

**Input:** a set of the expert labels $\mathcal{E}$, and a set of ballots $\phi_2$
**Output:** the delegation result $\delta$

**Init:**
- For $i \in [1, m]$, create and initiate $D_i = 0$.

**Delegation interpretation:**
- For each ballot $B \in \phi_2$: parse $B$ in form of $(\mathsf{V}_j, \textsc{Cast}, v_j, \alpha_j)$; if $v_j = (\mathsf{Delegate}, \mathsf{E}_i)$ for some $\mathsf{E}_i \in \mathcal{E}$, then $D_i := D_i + \alpha_j$.

**Output:**
- Return $\delta := \{(\mathsf{E}_i, D_i)\}_{i \in [m]}$.

Fig. 4: The delegation calculation algorithm DelCal

Since sending data to the blockchain consumes coins, we implicitly assume all the experts $\mathcal{E}$ and voters $\mathcal{V}$ have spare coins to pay the transaction fees that is incurred during the protocol execution. More specifically, we let each party prepare $\{\mathsf{In}_i\}_{i=1}^{\ell_1}, \{\mathsf{Out}_j\}_{j=1}^{\ell_2}$ s.t.

$$\sum_{i=1}^{\ell_1} \mathsf{In}_i.\mathsf{Value} \geq \sum_{j=1}^{\ell_2} \mathsf{Out}_j.\mathsf{Value} \ .$$

Denote the corresponding coins owned by a voter $\mathsf{V}_i \in \mathcal{V}$, an expert $\mathsf{E}_j \in \mathcal{E}$, and a voting committee member $\mathsf{C}_t \in \mathcal{C}$ as $(\{\mathsf{In}_\eta^{(\mathsf{V}_i)}\}_{\eta=1}^{\ell_1}, \{\mathsf{Out}_\eta^{(\mathsf{V}_i)}\}_{\eta=1}^{\ell_2})$, $(\{\mathsf{In}_\eta^{(\mathsf{E}_j)}\}_{\eta=1}^{\ell_1}, \{\mathsf{Out}_\eta^{(\mathsf{E}_j)}\}_{\eta=1}^{\ell_2})$, and $(\{\mathsf{In}_\eta^{(\mathsf{C}_t)}\}_{\eta=1}^{\ell_1}, \{\mathsf{Out}_\eta^{(\mathsf{C}_t)}\}_{\eta=1}^{\ell_2})$, respectively. The protocol is depicted in Fig. **??**. It consists of preparation phase, voting/delegation phase, and tally phase.

**Sending/Reading data to/from $\mathcal{F}_{\text{LEDGER}}$.** Fig. **??** describes the macro for a party to send and read data to/from the blockchain $\mathcal{F}_{\text{LEDGER}}$. According the blockchain model proposed by [**?**], three types of delays need to be considered. First, we have a bounded network delay, and it is assumed that all messages can be delivered within $\Delta_1$ rounds, which is $2\Delta_1$ clock-ticks in [**?**]. Subsequently, a desynchronised user can get up-to-date within $2\Delta_1$ rounds (i.e. $4\Delta_1$ clock-ticks) after registration. The second type of delay is the fact that the adversary can hold a valid transaction up to certain blocks, but she cannot permanently deny service to (or DoS) such a transaction. This is modeled by the ExtendPolicy in $\mathcal{F}_{\text{LEDGER}}$, where if a transaction is more than $\Delta_2$ rounds (i.e. $2\Delta_2$ clock-ticks) old, and is still valid with respect to the current state, then it will be included into the state. Finally, we have a so-called windowsize. Namely, the adversary can set state-slackness of all the honest parties up to the windowsize, which is consistent with the *common prefix* property in [**?**]. Hence, all the honest parties can have a common state of any blocks that have been

**The tally algorithm TallyAlg**

**Input:** a set of the voters $\mathcal{V}$, a set of the experts $\mathcal{E}$, two sets of ballots $\phi_1, \phi_2$ and the delegation $\delta$.
**Output:** the tally result $\tau$

**Init:**

- Create and initiate $\tau_{\mathsf{yes}} = 0$, $\tau_{\mathsf{no}} = 0$ and $\tau_{\mathsf{abstain}} = 0$.
- Parse $\delta$ as $\{(\mathsf{E}_i, D_i)\}_{i\in[m]}$.

**Tally Computation:**

- For each ballot $B \in \phi_2$: parse $B$ in form of $(\mathsf{V}_j, \text{CAST}, v_j, \alpha_j)$; if $v_j = (\mathsf{Vote}, a_j)$ for some $a_j \in \{\mathsf{yes}, \mathsf{no}, \mathsf{abstain}\}$, then $\tau_{a_j} := \tau_{a_j} + \alpha_j$.
- For each ballot $B \in \phi_1$: parse $B$ in form of $(\mathsf{E}_i, \text{VOTE}, b_i)$ for some $b_i \in \{\mathsf{yes}, \mathsf{no}, \mathsf{abstain}\}$, then $\tau_{b_i} := \tau_{b_i} + D_i$.

**Output:**

- Return $\tau := (\tau_{\mathsf{yes}}, \tau_{\mathsf{no}}, \tau_{\mathsf{abstain}})$.

Fig. 5: The tally algorithm TallyAlg

proposed more than windowsize. Denote $\Delta_3$ rounds (i.e. $2\Delta_3$ clock-ticks) as the windowsize.

To send a message $x$ to $\mathcal{F}_{\text{LEDGER}}$, we need to first check if this party has deregistered and desynchronized. If so, the party needs to first send $(\text{REGISTER}, \mathsf{sid})$ to $\mathcal{F}_{\text{LEDGER}}$. Note that the registered but desynchronized party can still send a transaction before it is fully updated. We simply make a 'dummy' transaction whose input coins and output coins share the same owner (spending condition), and the message $x$ is stored in the payload of the transaction. To read a message (stored in the payload of some transaction) from $\mathcal{F}_{\text{LEDGER}}$, analogously a deregistered party needs to first send $(\text{REGISTER}, \mathsf{sid})$ to $\mathcal{F}_{\text{LEDGER}}$. After $4\delta_1$ clock-ticks, the party can get synchronised. In order to receive the latest message, the party needs to wait a maximum of $2(\Delta_2 + \Delta_3)$ clock-ticks for the transaction that carries the intended message to be included in the state of the party.

## V. A NEW UNIT VECTOR ZK PROOF

**Zero-knowledge proofs/arguments.** Let $\mathcal{L}$ be an NP language and $\mathcal{R}_\mathcal{L}$ is its corresponding polynomial time decidable binary relation, i.e., $\mathcal{L} := \{x \mid \exists w : (x, w) \in \mathcal{R}_\mathcal{L}\}$. We say a statement $x \in \mathcal{L}$ if there is a witness $w$ such that $(x, w) \in \mathcal{R}_\mathcal{L}$. Let the prover $P$ and the verifier $V$ be two PPT interactive algorithms. Denote $\tau \leftarrow \langle P(x, w), V(x)\rangle$ as the public transcript produced by $P$ and $V$. After the protocol, $V$ accepts the proof if and only if $\phi(x, \tau) = 1$, where $\phi$ is a public predicate function.

*Definition 2:* We say $(P, V)$ is a perfectly complete proof/argument for an NP relation $\mathcal{R}_\mathcal{L}$ if for all non-uniform PPT interactive adversaries $\mathcal{A}$ it satisfies

- Perfect completeness:
$$\Pr\left[\begin{array}{l}(x, w) \leftarrow \mathcal{A}; \tau \leftarrow \langle P(x, w), V(x)\rangle : \\ (x, w) \in \mathcal{R}_\mathcal{L} \vee \phi(x, \tau) = 1\end{array}\right] = 1$$
- (Computational) soundness:
$$\Pr\left[\begin{array}{l}x \leftarrow \mathcal{A}; \tau \leftarrow \langle \mathcal{A}, V(x)\rangle : \\ x \notin \mathcal{L} \wedge \phi(x, \tau) = 1\end{array}\right] = \mathsf{negl}(\lambda)$$

Let $V(x; r)$ denote the verifier $V$ is executed on input $x$ with random coin $r$. A proof/argument $(P, V)$ is called *public coin* if the verifier $V$ picks his challenges randomly and independently of the messages sent by the prover $P$.

*Definition 3:* We say a public coin proof/argument $(P, V)$ is a perfect special honest verifier zero-knowledge (SHVZK) for a NP relation $\mathcal{R}_\mathcal{L}$ if there exists a PPT simulator $\mathsf{Sim}$ such that
$$\Pr\left[\begin{array}{l}(x, w, r) \leftarrow \mathcal{A}; \\ \tau \leftarrow \langle P(x, w), V(x; r)\rangle : \\ (x, w) \in \mathcal{R}_\mathcal{L} \wedge \\ \wedge\ \mathcal{A}(\tau) = 1\end{array}\right] \approx \Pr\left[\begin{array}{l}(x, w, r) \leftarrow \mathcal{A}; \\ \tau \leftarrow \mathsf{Sim}(x; r) : \\ (x, w) \in \mathcal{R}_\mathcal{L} \wedge \\ \wedge\ \mathcal{A}(\tau) = 1\end{array}\right]$$

Public coin SHVZK proofs/arguments can be transformed to a non-interactive one (in the random oracle model [**?**]) by using Fiat-Shamir heuristic [**?**] where a cryptographic hash function is used to compute the challenge instead of having an online verifier.

**Schwartz-Zippel lemma.** For completeness, we recap a variation of the Schwartz-Zippel lemma [**?**] that will be used in proving the soundness of the zero-knowledge protocols.

*Lemma 1 (Schwartz-Zippel):* Let $f$ be a non-zero multivariate polynomial of degree $d$ over $\mathbb{Z}_p$, then the probability of $f(x_1, \ldots, x_n) = 0$ evaluated with random $x_1, \ldots, x_n \leftarrow \mathbb{Z}_p$ is at most $\frac{d}{p}$.

Therefore, there are two multi-variate polynomials $f_1, f_2$. If $f_1(x_1, \ldots, x_n) - f_2(x_1, \ldots, x_n) = 0$ for random $x_1, \ldots, x_n \leftarrow \mathbb{Z}_p$, then we can assume that $f_1 = f_2$. This is because, if $f_1 \neq f_2$, the probability that the above equation holds is bounded by $\frac{\max(d_1, d_2)}{p}$, which is negligible in $\lambda$.

**Pedersen commitment.** In the unit vector zero-knowledge proof, we use Pedersen commitment as a building block. It is perfectly hiding and computationally binding under the discrete logarithm assumption. More specifically, it consists of the following 4 PPT algorithms. Note that those algorithms (implicitly) take as input the same group parameters, $\mathsf{param} \leftarrow \mathsf{Gen}^{\mathsf{gp}}(1^\lambda)$.

- $\mathsf{KeyGen}^{\mathsf{C}}(\mathsf{param})$: pick $s \leftarrow \mathbb{Z}_q^*$ and set $\mathsf{ck} := h = g^s$, and output $\mathsf{ck}$.
- $\mathsf{Com}_{\mathsf{ck}}(m; r)$: output $c := g^m h^r$ and $d := (m, r)$.
- $\mathsf{Open}(c, d)$: output $d := (m, r)$.
- $\mathsf{Verify}_{\mathsf{ck}}(c, d)$: return $\mathsf{valid}$ if and only if $c = g^m h^r$.

Pedersen commitment is also additively homomorphic, i.e.
$$\mathsf{Com}_{\mathsf{ck}}(m_1; r_1) \cdot \mathsf{Com}_{\mathsf{ck}}(m_2; r_2) = \mathsf{Com}_{\mathsf{ck}}(m_1 + m_2; r_1 + r_2)\ .$$

**The proposed unit vector ZK proof/argument.** We denote a unit vector of length $n$ as $\mathbf{e}_i^{(n)} = (e_{i,0}, \ldots, e_{i,n-1})$, where its $i$-th coordinate is 1 and the rest coordinates are 0. Conventionally, to show a vector of ElGamal ciphertexts element-wise encrypt a unit vector, Chaum-Pedersen proofs [**?**] are used to show each of the ciphertexts encrypts either 0 or 1 (via Sigma OR composition) and the product of all the ciphertexts encrypts 1. Such kind of proof is used in many well-known voting

### Sending and reading messages

**Macro** $\mathsf{Send\text{-}Msg}(x, \{\mathsf{In}_i\}_{i=1}^{\ell_1}, \{\mathsf{Out}_j\}_{j=1}^{\ell_2})$**:**

- If the party has deregistered and desynchronized:
  - Send $(\text{REGISTER}, \mathsf{sid})$ to $\mathcal{F}_{\text{LEDGER}}$.
  - Send $\Big(\text{SUBMIT}, sid, \mathsf{Tx}(\{\mathsf{In}_i\}_{i=1}^{\ell_1}; \{\mathsf{Out}_j\}_{j=1}^{\ell_2}; x)\Big)$ to $\mathcal{F}_{\text{LEDGER}}$.
  - Send $(\text{DE-REGISTER}, \mathsf{sid})$ to $\mathcal{F}_{\text{LEDGER}}$.
- If the party is already synchronized:
  - Send $\Big(\text{SUBMIT}, sid, \mathsf{Tx}(\{\mathsf{In}_i\}_{i=1}^{\ell_1}; \{\mathsf{Out}_j\}_{j=1}^{\ell_2}; x)\Big)$ to $\mathcal{F}_{\text{LEDGER}}$.

**Macro** $\mathsf{Read\text{-}Msg}$**:**

- If the party has deregistered and desynchronized:
  - Send $(\text{REGISTER}, \mathsf{sid})$ to $\mathcal{F}_{\text{LEDGER}}$.
  - Wait for $\max\{4\Delta_1, 2(\Delta_2 + \Delta_3)\}$ clock-ticks by keeping sending $(\text{TICK}, \mathsf{sid})$ to the $\mathcal{G}_{\text{CLOCK}}$.
  - Send $(\text{READ}, sid)$ to $\mathcal{F}_{\text{LEDGER}}$ and receive $(\text{READ}, sid, \mathsf{data})$ from $\mathcal{F}_{\text{LEDGER}}$.
  - Send $(\text{DE-REGISTER}, \mathsf{sid})$ to $\mathcal{F}_{\text{LEDGER}}$.
- If the party is already synchronized:
  - Wait for $\max\{4\Delta_1, 2(\Delta_2 + \Delta_3)\}$ clock-ticks by keeping sending $(\text{TICK}, \mathsf{sid})$ to the $\mathcal{G}_{\text{CLOCK}}$.
  - Send $(\text{READ}, sid)$ to $\mathcal{F}_{\text{LEDGER}}$ and receive $(\text{READ}, sid, \mathsf{data})$ from $\mathcal{F}_{\text{LEDGER}}$.
- Return $\mathsf{data}$.

Fig. 6: Macro for sending and receiving message via $\mathcal{F}_{\text{LEDGER}}$

### The voting protocol $\Pi_{\text{VOTE}}^{t,k,m,n}$

Denote the corresponding coins owned by a voter $\mathsf{V}_i \in \mathcal{V}$, an expert $\mathsf{E}_j \in \mathcal{E}$, and a voting committee member $\mathsf{C}_t \in \mathcal{C}$ as $(\{\mathsf{In}_\eta^{(\mathsf{V}_i)}\}_{\eta=1}^{\ell_1}, \{\mathsf{Out}_\eta^{(\mathsf{V}_i)}\}_{\eta=1}^{\ell_2})$, $(\{\mathsf{In}_\eta^{(\mathsf{E}_j)}\}_{\eta=1}^{\ell_1}, \{\mathsf{Out}_\eta^{(\mathsf{E}_j)}\}_{\eta=1}^{\ell_2})$, and $(\{\mathsf{In}_\eta^{(\mathsf{C}_t)}\}_{\eta=1}^{\ell_1}, \{\mathsf{Out}_\eta^{(\mathsf{C}_t)}\}_{\eta=1}^{\ell_2})$, respectively.

**Preparation phase:**

- Upon receiving $(\text{INIT}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the committee $\mathsf{C}_j$, $j \in [k]$ sends $(\text{KEYGEN}, \mathsf{sid})$ to $\mathcal{F}_{\text{DKG}}^{t,k}$ to generate $\mathsf{pk}$.

**Voting/Delegation phase:**

- Upon receiving $(\text{VOTE}, \mathsf{sid}, v_j)$ from the environment $\mathcal{Z}$, the expert $\mathsf{E}_j$, $j \in [m]$ does the following:
  - Send $(\text{READPK}, \mathsf{sid})$ to $\mathcal{F}_{\text{DKG}}^{t,k}$, and receive $(\text{PUBLICKEY}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{F}_{\text{DKG}}^{t,k}$.
  - Set the unit vector $\mathbf{e}^{(3)} \leftarrow \mathsf{encode}^{\mathsf{E}}(v_j)$. Compute $\mathbf{c_j}^{(3)} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{e}^{(3)})$ and its NIZK proof $\pi_j$ (Cf. Sec. **??**).
  - Execute macro $\mathsf{Send\text{-}Msg}\Big((\mathbf{c_j}^{(3)}, \pi_j), \{\mathsf{In}_\eta^{(\mathsf{E}_j)}\}_{\eta=1}^{\ell_1}, \{\mathsf{Out}_\eta^{(\mathsf{E}_j)}\}_{\eta=1}^{\ell_2}\Big)$. (Cf. Fig. **??** )
- Upon receiving $(\text{CAST}, \mathsf{sid}, v_i, \alpha_i)$ from the environment $\mathcal{Z}$, the voter $\mathsf{V}_i$, $i \in [n]$ does the following:
  - Send $(\text{READPK}, \mathsf{sid})$ to $\mathcal{F}_{\text{DKG}}^{t,k}$, and receive $(\text{PUBLICKEY}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{F}_{\text{DKG}}^{t,k}$.
  - Set the unit vector $\mathbf{e}^{(m+3)} \leftarrow \mathsf{encode}^{\mathsf{V}}(v_i)$. Compute $\mathbf{u_i}^{(m+3)} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{e}^{(m+3)})$ and its NIZK proof $\sigma_i$ (Cf. Sec. **??**).
  - Execute macro $\mathsf{Send\text{-}Msg}\Big((\mathbf{u_i}^{(m+3)}, \sigma_i, \alpha_i), \{\mathsf{In}_\eta^{(\mathsf{V}_i)}\}_{\eta=1}^{\ell_1}, \{\mathsf{Out}_\eta^{(\mathsf{V}_i)}\}_{\eta=1}^{\ell_2}\Big)$. (Cf. Fig. **??** )

**Tally phase:**

- Upon receiving $(\text{DELCAL}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the committee $\mathsf{C}_t$, $t \in [k]$ does:
  - Execute macro $\mathsf{Read\text{-}Msg}$ and obtain $\mathsf{data}$.
  - Fetch the ballots $\{(\mathbf{c_i}^{(3)}, \pi_i)\}_{i\in[m]}$ and $\{(\mathbf{u_j}^{(m+3)}, \sigma_j, \alpha_j)\}_{j\in[n]}$ from $\mathsf{data}$.
  - For $i \in [m]$, check $\mathsf{Verify}(\mathbf{c_i}^{(3)}, \pi_i) = 1$; for $j \in [n]$, $\mathsf{Verify}(\mathbf{u_j}^{(m+3)}, \sigma_j) = 1$. Remove all the invalid ballots.
  - For $j \in [n]$, if a valid $\mathbf{u_j}^{(m+3)}$ is posted, parse $\mathbf{u_j}^{(m+3)}$ to $(\mathbf{a_j}^{(m)}, \mathbf{b_j}^{(3)})$.
  - For $j \in [n]$, $\ell \in [0, m-1]$, compute $z_{i,\ell} := a_{j,\ell}^{\alpha_j}$.
  - For $i \in [0, m-1]$, compute $s_i := \prod_{\ell=1}^{n} z_{\ell,i}$ and jointly decrypt it to $w_i$ (Cf. [**?**]).
- Upon receiving $(\text{TALLY}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the committee $\mathsf{C}_t$, $t \in [k]$ does:
  - For $i \in [0, m-1]$, $\ell \in [0, 2]$, compute $d_{i,\ell} := c_{i,\ell}^{w_i}$.
  - For $\ell \in [0, 2]$, compute $x_\ell := \prod_{j=0}^{m-1} d_{j,\ell} \cdot \prod_{j=1}^{n} b_{j,\ell}$ and jointly decrypt it to $y_\ell$ (Cf. [**?**]). Execute macro $\mathsf{Send\text{-}Msg}\Big((x_\ell, y_\ell), \{\mathsf{In}_\eta^{(\mathsf{C}_t)}\}_{\eta=1}^{\ell_1}, \{\mathsf{Out}_\eta^{(\mathsf{C}_t)}\}_{\eta=1}^{\ell_2}\Big)$. (Cf. Fig. **??** )
- Upon receiving $(\text{READTALLY}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the party $P$ does the following:
  - Execute macro $\mathsf{Read\text{-}Msg}$ and obtain $\mathsf{data}$.
  - Fetch $\{(x_i, y_i)\}_{i\in[0,2]}$ from $\mathsf{data}$, and return $(\text{READTALLYRETURN}, \mathsf{sid}, (y_0, y_1, y_2))$ to the environment $\mathcal{Z}$.

Fig. 7: The voting protocol $\Pi_{\text{VOTE}}^{t,k,m,n}$ in $\{\mathcal{F}_{\text{LEDGER}}, \mathcal{F}_{\text{DKG}}^{t,k}\}$-hybrid model

**The algorithm that maps $i \in [0, n-1]$ to $\mathbf{e}_i^{(n)}$**

**Input:** index $i = (i_1, \ldots, i_{\log n}) \in \{0,1\}^{\log n}$
**Output:** unit vector $\mathbf{e}_i^{(n)} = (e_{i,0}, \ldots, e_{i,n-1}) \in \{0,1\}^n$
1. For $\ell \in [\log n]$, set $b_{\ell,0} := 1 - i_\ell$ and $b_{\ell,1} := i_\ell$;
2. For $j \in [0, n-1]$, set $e_{i,j} := \prod_{\ell=1}^{\log n} b_{\ell, j_\ell}$, where $j_1, \ldots, j_{\log n}$ is the binary representation of $j$;
3. Return $\mathbf{e}_i^{(n)} = (e_{i,0}, \ldots, e_{i,n-1})$;

Fig. 8: The algorithm that maps $i \in [0, n-1]$ to $\mathbf{e}_i^{(n)}$

schemes, e.g., Helios. However, the proof size is linear in the length of the unit vector, and thus the communication overhead is quite significant when the unit vector length becomes larger.

In this section, we propose a novel special honest verifier ZK (SHVZK) proof for unit vector that allows the prover to convince the verifier that a vector of ciphertexts $(C_0, \ldots, C_{n-1})$ encrypts a unit vector $\mathbf{e}_i^{(n)}$, $i \in [0, n-1]$ with $O(\log n)$ proof size. Without loss of generality, assume $n$ is a perfect power of 2. If not, we append $\mathsf{Enc}_{\mathsf{pk}}(0;0)$ (i.e., trivial ciphertexts) to make the total number of ciphertexts to be the next power of 2. The proposed SHVZK protocol can also be Fiat-Shamir transformed to a non-interactive ZK (NIZK) proof in the random oracle model. The basic idea of our construction is inspired by [**?**], where Groth and Kohlweiss proposed a Sigma protocol for the prover to show that he knows how to open one out of many commitments. The key idea behind our construction is that there exists a data-oblivious algorithm that can take input as $i \in \{0,1\}^{\log n}$ and output the unit vector $\mathbf{e}_i^{(n)}$. Let $i_1, \ldots, i_{\log n}$ be the binary representation of $i$. The algorithm is depicted in Fig. **??**.

Intuitively, we let the prover first bit-wisely commit the binary presentation of $i \in [0, n-1]$ for the unit vector $\mathbf{e}_i^{(n)}$. The prover then shows that each of the commitments of $(i_1, \ldots, i_{\log n})$ indeed contain 0 or 1, using the Sigma protocol proposed in Section 2.3 of [**?**]. Note that in the 3rd move of such a Sigma protocol, the prover reveals a degree-1 polynomial of the committed message. Denote $z_{\ell,1} := i_\ell x + \beta_\ell$, $\ell \in [\log n]$ as the corresponding degree-1 polynomials, where $\beta_\ell$ are chosen by the prover and $x$ is chosen by the verifier. By linearity, we can also define $z_{\ell,0} := x - z_{\ell,1} = (1 - i_\ell)x - \beta_\ell$, $\ell \in [\log n]$. According to the algorithm described in Fig.**??**, for $j \in [0, n-1]$, let $j_1, \ldots, j_{\log n}$ be the binary representation of $j$, and the product $\prod_{\ell=1}^{\log n} z_{\ell, j_\ell}$ can be viewed as a degree-$(\log n)$ polynomial of the form

$$p_j(x) = e_{i,j} x^{\log n} + \sum_{k=0}^{\log n - 1} p_{j,k} x^k$$

for some $p_{j,k}$, $k \in [0, \log n - 1]$. We then use batch verification to show that each of $C_j$ indeed encrypts $e_{i,j}$. More specifically, for a randomly chosen $y \leftarrow \mathbb{Z}_p$, let $E_j := (C_j)^{x^{\log n}} \cdot \mathsf{Enc}(-p_j(x); 0)$; the prover needs to show that $E := \prod_{j=0}^{n-1} (E_j)^{y^j} \cdot \prod_{k=0}^{\log n - 1} (D_k)^{x^k}$ encrypts 0, where $D_\ell := \mathsf{Enc}_{\mathsf{pk}}(\sum_{j=0}^{n-1} (p_{j,\ell} \cdot y^j); R_\ell)$, $\ell \in [0, \log n - 1]$ with fresh randomness $R_\ell \in \mathbb{Z}_p$. The construction is depicted in Fig. **??**, and it consists of 5 moves. Both the prover and the verifier shares a common reference string (CRS), which is a Pedersen commitment key that can be generated using random oracle. The prover first commits to each bits of the binary representation of $i$, and the commitments are denoted as $I_\ell$, $\ell \in [\log n]$. Subsequently, it produces $B_\ell, A_\ell$ as the first move of the Sigma protocol in Sec. 2.3 of [**?**] showing $I_\ell$ commits to 0 or 1. Jumping ahead, later the prover will receive a challenge $x \leftarrow \{0,1\}^\lambda$, and it then computes the third move of the Sigma protocols by producing $\{z_\ell, w_\ell, v_\ell\}_{\ell=1}^{\log n}$. To enable batch verification, before that, the prover is given another challenge $y \leftarrow \{0,1\}^\lambda$ in the second move. The prover computes and sends $\{D_\ell\}_{\ell=0}^{\log n - 1}$. The verification consists of two parts. In the first part, the verifier checks the following equations to ensure that $I_\ell$ commits to 0 or 1.

- $(I_\ell)^x \cdot B_\ell = \mathsf{Com}_{\mathsf{ck}}(z_\ell; w_\ell)$
- $(I_\ell)^{x - z_\ell} \cdot A_\ell = \mathsf{Com}_{\mathsf{ck}}(0; v_\ell)$

In the second part, the verifier checks if

$$\prod_{j=0}^{n-1} \left((C_j)^{x^{\log n}} \cdot \mathsf{Enc}_{\mathsf{pk}}(-\prod_{\ell=1}^{\log n} z_{\ell, j_\ell}; 0)\right)^{y^j} \cdot \prod_{\ell=0}^{\log n - 1} (D_\ell)^{x^\ell}$$

is encryption of 0 by asking the prover to reveal the randomness.

*Theorem 1:* The protocol described in Fig. **??** is a 5-move public coin special honest verifier zero-knowledge argument of knowledge of $\mathbf{e}_i^{(n)} = (e_{i,0}, \ldots, e_{i,n-1}) \in \{0,1\}^n$ and $(r_0, \ldots, r_{n-1}) \in (\mathbb{Z}_p)^n$ such that $C_j = \mathsf{Enc}_{\mathsf{pk}}(e_{i,j}; r_j)$, $j \in [0, n-1]$ under the DDH assumption.

*Proof:* For perfect completeness, we first observe that the verification equations $(I_\ell)^x \cdot B_\ell = \mathsf{Com}_{\mathsf{ck}}(z_\ell; w_\ell)$ and $(I_\ell)^{x - z_\ell} \cdot A_\ell = \mathsf{Com}_{\mathsf{ck}}(0; v_\ell)$ holds. Indeed, by additively homomorphic property of the commitment scheme, $(I_\ell)^x \cdot B_\ell = \mathsf{Com}_{\mathsf{ck}}(i_\ell \cdot x + \beta_\ell; \alpha_\ell \cdot x + \gamma_\ell)$ and $(I_\ell)^{x - z_\ell} \cdot A_\ell = \mathsf{Com}_{\mathsf{ck}}(i_\ell \cdot (x - z_\ell) + i_\ell \cdot \beta_\ell; \alpha_\ell \cdot (x - z_\ell) + \delta_\ell) = \mathsf{Com}_{\mathsf{ck}}(i_\ell (1 - i_\ell) \cdot x; v_\ell)$. Since $i_\ell(1 - i_\ell) = 0$ when $i_\ell \in \{0,1\}$, we have $(I_\ell)^{x - z_\ell} \cdot A_\ell = \mathsf{Com}_{\mathsf{ck}}(0; v_\ell)$. Moreover, for each $j \in [0, n-1]$, $\prod_{\ell=1}^{\log n} z_{\ell, j_\ell}$ is a polynomial in the form of

$$p_j(x) = e_{i,j} x^{\log n} + \sum_{k=0}^{\log n - 1} p_{j,k} x^k$$

where $x$ is the verifier's challenge. Therefore, it is easy to see

$$\begin{aligned}
&\prod_{j=0}^{n-1} \left((C_j)^{x^{\log n}} \cdot \mathsf{Enc}_{\mathsf{pk}}(-\prod_{\ell=1}^{\log n} z_{\ell, j_\ell}; 0)\right)^{y^j} \\
&\cdot \prod_{\ell=0}^{\log n - 1} \mathsf{Enc}_{\mathsf{pk}}(\sum_{j=0}^{n-1} (p_{j,\ell} \cdot y^j); R_\ell)^{x^\ell} \\
&= \mathsf{Enc}_{\mathsf{pk}}\Big(\sum_{j=0}^{n-1} \big(e_{i,j} \cdot x^{\log n} - p_j(x) + \sum_{\ell=0}^{\log n - 1} p_{j,\ell} \cdot x^\ell\big) \cdot y^j; R\Big) \\
&= \mathsf{Enc}_{\mathsf{pk}}(0; R) \ .
\end{aligned}$$

For soundness, first of all, the Sigma protocols for commitments of $i_\ell$, $\ell \in [\log n]$ is specially sound, i.e., given two transactions with the same $\{I_\ell, B_\ell, A_\ell\}_{\ell=1}^{\log n}$ and two different $x$ and $\{z_\ell, w_\ell, v_\ell\}_{\ell=1}^{\log n}$, there exists a PPT extractor that can output the corresponding witness $i_\ell \in \{0,1\}$.

**Unit vector ZK argument**

**CRS:** the commitment key $\mathsf{ck}$
**Statement:** the public key $\mathsf{pk}$ and the ciphertexts $C_0 := \mathsf{Enc}_{\mathsf{pk}}(e_{i,0}; r_0), \ldots, C_{n-1} := \mathsf{Enc}_{\mathsf{pk}}(e_{i,n-1}; r_{n-1})$
**Witness:** the unit vector $\mathbf{e}_i^{(n)} \in \{0,1\}^n$ and the randomness $r_0, \ldots, r_{n-1} \in \mathbb{Z}_p$
**Protocol:**

- The prover $P$, for $\ell = 1, \ldots, \log n$, do:
  - Pick random $\alpha_\ell, \beta_\ell, \gamma_\ell, \delta_\ell \leftarrow \mathbb{Z}_p$;
  - Compute $I_\ell := \mathsf{Com}_{\mathsf{ck}}(i_\ell; \alpha_\ell)$, $B_\ell := \mathsf{Com}_{\mathsf{ck}}(\beta_\ell; \gamma_\ell)$ and $A_\ell := \mathsf{Com}_{\mathsf{ck}}(i_\ell \cdot \beta_\ell; \delta_\ell)$;
- $P \to V$: $\{I_\ell, B_\ell, A_\ell\}_{\ell=1}^{\log n}$;
- $V \to P$: Random $y \leftarrow \{0,1\}^\lambda$;
- The prover $P$ for $\ell = 0, \ldots, \log n - 1$, do:
  - Pick random $R_\ell \leftarrow \mathbb{Z}_p$ and compute $D_\ell := \mathsf{Enc}_{\mathsf{pk}}\big(\sum_{j=0}^{n-1}(p_{j,\ell} \cdot y^j); R_\ell\big)$
- $P \to V$: $\{D_\ell\}_{\ell=0}^{\log n - 1}$;
- $V \to P$: Random $x \leftarrow \{0,1\}^\lambda$;
- The prover $P$ does the following:
  - Compute $R := \sum_{j=0}^{n-1}(r_j \cdot x^{\log n} \cdot y^j) + \sum_{\ell=0}^{\log n - 1}(R_\ell \cdot x^\ell)$;
  - For $\ell = 1, \ldots, \log n$, compute $z_\ell := i_\ell \cdot x + \beta_\ell$, $w_\ell := \alpha_\ell \cdot x + \gamma_\ell$, and $v_\ell := \alpha_\ell(x - z_\ell) + \delta_\ell$;
- $P \to V$: $R$ and $\{z_\ell, w_\ell, v_\ell\}_{\ell=1}^{\log n}$

**Verification:**

- Check the followings:
- For $\ell = 1, \ldots, \log n$, do:
  - $(I_\ell)^x \cdot B_\ell = \mathsf{Com}_{\mathsf{ck}}(z_\ell; w_\ell)$
  - $(I_\ell)^{x - z_\ell} \cdot A_\ell = \mathsf{Com}_{\mathsf{ck}}(0; v_\ell)$
- $\prod_{j=0}^{n-1}\big((C_j)^{x^{\log n}} \cdot \mathsf{Enc}_{\mathsf{pk}}(-\prod_{\ell=1}^{\log n} z_{\ell,j_\ell}; 0)\big)^{y^j} \cdot \prod_{\ell=0}^{\log n - 1}(D_\ell)^{x^\ell} = \mathsf{Enc}_{\mathsf{pk}}(0; R)$, where $z_{j,1} = z_j$ and $z_{j,0} = x - z_j$.

Fig. 9: Unit vector ZK argument

Moreover, $\prod_{j=0}^{n-1}\big((C_j)^{x^{\log n}} \cdot \mathsf{Enc}_{\mathsf{pk}}(-\prod_{\ell=1}^{\log n} z_{\ell,j_\ell}; 0)\big)^{y^j}$ builds a degree-$\log n$ polynomial w.r.t. $x$ in the plaintext. While, $\prod_{\ell=0}^{\log n-1}(D_\ell)^{x^\ell}$ encrypts a degree-$(\log n - 1)$ polynomial w.r.t. $x$. Since $x$ is randomly sampled after $D_\ell$ is committed, Schwartz-Zippel lemma, $\prod_{j=0}^{n-1}\big((C_j)^{x^{\log n}} \cdot \mathsf{Enc}_{\mathsf{pk}}(-\prod_{\ell=1}^{\log n} z_{\ell,j_\ell}; 0)\big)^{y^j} \cdot \prod_{\ell=0}^{\log n-1}(D_\ell)^{x^\ell}$ encrypts a zero polynomial w.r.t. $x$ with overwhelming probability if the polynomial evaluation is 0. Therefore, $Q(y) := \sum_{j=0}^{n-1}(e_{i,j} - \prod_{\ell=1}^{\log n} i_{\ell,j_\ell}) \cdot y^j = 0$ with overwhelming probability. Similarly, by Schwartz-Zippel lemma, $Q(y)$ is a zero polynomial; hence, we have for $j \in [0, n-1]$, $e_{i,j} = \prod_{\ell=1}^{\log n} i_{\ell,j_\ell}$ with overwhelming probability.

In terms of special honest verifier zero-knowledge, we now construct a simulator $\mathsf{Sim}$ that takes input as the statement $(C_0, \ldots, C_{n-1})$ and the given challenges $x, y \in \{0,1\}^\lambda$, and it outputs a simulated transcript whose distribution is indistinguishable from the real one. More specifically, $\mathsf{Sim}$ first randomly picks $i_\ell \leftarrow \{0,1\}$ and $\alpha_\ell, \beta_\ell, \gamma_\ell, \delta_\ell \leftarrow \mathbb{Z}_p$, $\ell \in [\log n]$. It then computes $\{I_\ell, B_\ell, A_\ell\}_{\ell=1}^{\log n}$ and $\{z_\ell, w_\ell, v_\ell\}_{\ell=1}^{\log n}$ according to the protocol description. For $\ell \in \{1, \ldots, \log n - 1\}$, it then picks random $U_\ell, R_\ell \leftarrow \mathbb{Z}_p$ and computes $D_\ell := \mathsf{Enc}_{\mathsf{pk}}(U_\ell; R_\ell)$. It then randomly picks $R \leftarrow \mathbb{Z}_p$, computes

$$D_0 := \frac{\mathsf{Enc}_{\mathsf{pk}}(0; R)}{\prod_{j=0}^{n-1}\big((C_j)^{x^{\log n}} \mathsf{Enc}_{\mathsf{pk}}(-\prod_{\ell=1}^{\log n} z_{\ell,j_\ell}; 0)\big)^{y^j} \cdot \prod_{\ell=1}^{\log n-1}(D_\ell)^{x^\ell}}$$

After that, $\mathsf{Sim}$ outputs the simulated transcript as

$$\Big(\{I_\ell, B_\ell, A_\ell\}_{\ell=1}^{\log n}, y, \{D_\ell\}_{\ell=0}^{\log n-1}, x, \{z_\ell, w_\ell, v_\ell\}_{\ell=1}^{\log n}\Big) \; .$$

This concludes our proof. ∎

## VI. Security Ayalysis

The security of the treasury voting protocol is analysed in the UC framework. We provide Theorem **??** and its proof can be found in the full version.

*Theorem 2:* Let $k, n, m = \mathrm{poly}(\lambda)$ and $t > k/2$. Protocol $\Pi_{\text{VOTE}}^{t,k,n,m}$ described in Fig. **??** UC-realizes $\mathcal{F}_{\text{VOTE}}^{t,k,m,n}$ in the $\{\mathcal{F}_{\text{LEDGER}}, \mathcal{F}_{\text{DKG}}^{t,k}\}$-hybrid world against static corruption under the DDH assumption.

## VII. Implementation and performance

**Prototyping.** The proposed treasury system was implemented as a fully functional cryptocurrency prototype. As an underlying framework we used Scorex 2.0 [**?**] that provides basic blockchain functionality. It is a flexible modular framework designed particularly for fast prototyping with a rich set of already implemented functionalities such as asynchronous peer-to-peer network layer, built-in blockchain support with pluggable and extendable consensus module, simple transactions layer, JSON API for accessing the running node, etc. As treasury requires basic blockchain functions, we decided to select TwinsCoin [**?**] example and extend it with the proposed treasury system. Treasury integration required modification of the existed transactions structure and block validation rules, as well as introduction of new modules for keeping treasury state and managing transactions forging. All cryptographic protocols related to the voting procedure were implemented in a separate library to simplify code maintanance. It is also possible to reuse it not only in the blockchain systems but also as a standalone voting system. The implementation uses BouncyCastle library (ver.1.58) that provides needed elliptic curve math. Some operations in the finite field were

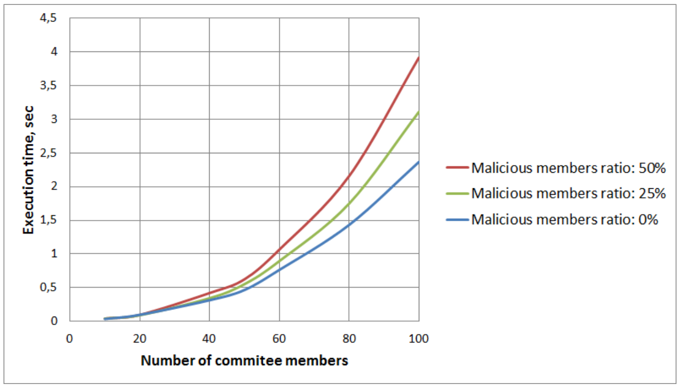


Fig. 10: DKG protocol execution time depending on the number of committee members

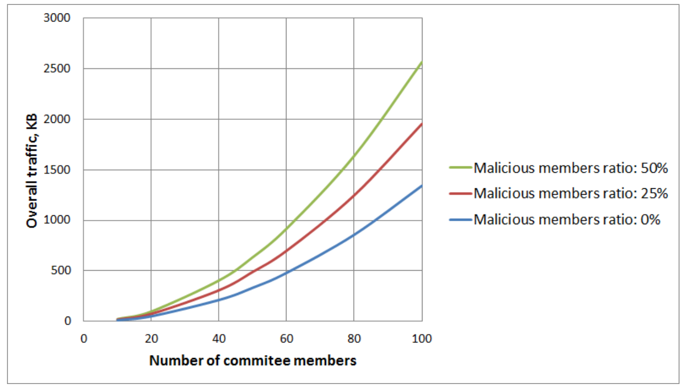


Fig. 11: Total size of the DKG protocol messages to be sent over the peer-to-peer network depending on the number of committee members

implemented with help of the BigInteger class from the Java Core. Subprotocols of the developed system were implemented exactly as they are described in the paper without any protocol-level optimizations.

**Test network.** For testing developed treasury prototype in real environment a local network of 12 full nodes was launched. It successfully worked for several days with dozens of epochs. The treasury network had 9 voters with different amount of stake, 3 experts, 12 candidates to the voting committee (10 of them were selected to participate). The numbers of proposals varied from 1 to 7. Treasury cycle had 780 blocks. Underlying blockchain with TwinsCoin consensus had block generation time of 10 seconds (or approximately 4.5 hours treasury cycle).

During the tests many abnormal situations were simulated, for instance, a malicious behavior of the committee members, absence of the voters and experts, refusal to participate in the decryption stage, etc. With a correctly working majority of the committee members, the voting results were always successfully obtained and rewards were correctly distributed.

**Evaluations.** For evaluating performance of the cryptographic protocols a special set of tests were developed as a part of the cryptographic library. The working station has Intel Core i7-6500U CPU @ 2.50GHz and 16GB RAM.

We benchmarked key generation protocol running time for different number of voting committee members: from 10 to 100 (high numbers might be required to guarantee honest majority on member random selection among large amount of members). Shared public key generation was made both for all honest committee members and in presence of malicious ones (any minority amount, their exact ratio does not have influence on protocol running time for any honest participant). Results are given in Fig. **??**.

Besides, there is an estimated amount of data needed to be transmitted over a P2P network to complete the protocol, in dependence of committee size and malicious members ratio. Results are given in Fig. **??** (recall that even controlling 50% of the committee, an attacker can break confidentiality of voters' ballots, but not their integrity or tally result).

Ballot generation is done once by a voter and takes less than 1 second for several hundreds of experts, so it has little influence on the voting protocol performance. To get tally results, it is needed to collect all ballots from participating voters, validate their correctness (via attached NIZK) and then do tally for all correct ballots. Figure **??** shows the prover's running time, the verifier's running time and the size of the unit vector ZK proof that has been used in the ballot casting.

Finally, the overall communication cost for all the voting ballots per project during the entire treasury period is depicted in Fig. **??**. In particular, for a treasury period with $5000$ voters and $50$ experts, the overall communication is approximately 20 MB per project.

**Remark.** Note that in practice, the treasury period is long enough, say, 30 days (approximately 4320 blocks for Bitcoin), so blockchain space overhead for treasury deployment in the cryptocurrency blockchain is insignificant. At the same time, we consider a sidechain approach [**?**], [**?**], [**?**] for treasury implementation as an effective solution. It allows separation of treasury functionality from the mainchain consensus, providing a number of advantages. In particular, treasury protocols do not influence the mainchain consensus, moving all implementation complexity in a sidechain. This modular construction also saves mainchain space for core clients.

## VIII. RELATED WORK

The Dash governance system (DGS) [**?**] also referred to as Dash governance by blockchain (DGBB) is the pioneer treasury implementation for cryptocurrency development funding on any real-world cryptocurrency. The DGS allows regular users on the Dash network to participate in the development process of the Dash cryptocurrency by allowing them submit project proposals (for advancing the cryptocurrency) to the network. A subset of users known as Masternodes then vote to decide what proposals from the submitted proposals get funding. Every voting cycle (approximately one month), winning proposals are voted for and funded from the accrued resources in the blockchain treasury. $10\%$ of all block rewards within each monthly voting period is contributed towards the blockchain treasury, from which proposals are then funded. Although the DGS works in practice, there are open questions to it. For instance, voting on the DGS is not private, thereby leaving nodes susceptible to coercion.

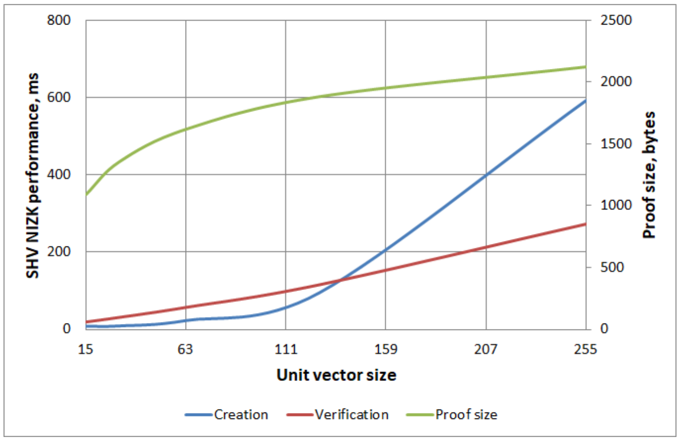


Fig. 12: The prover's running time, verifier's running time and the size of the unit vector ZK proof.

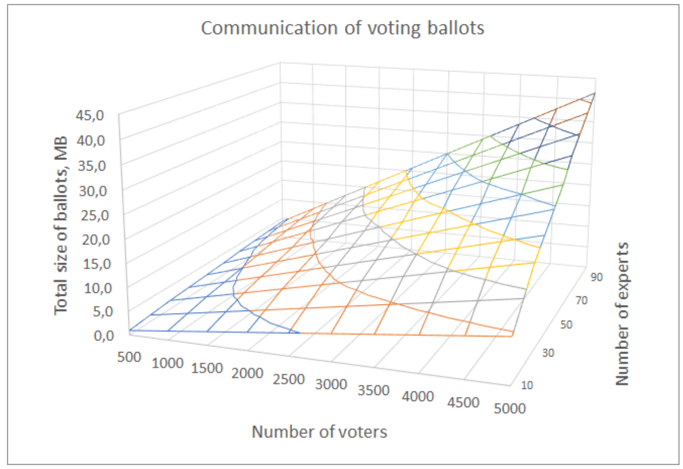


Fig. 13: The overall communication for all the voting ballots during an entire treasury period.

Beyond voting, the Dash Governance System (DGS) [?], [?], is the first self-sustenance/funding mechanism in any cryptocurrency or blockchain system. However, the DGS does not support delegative voting and ballot privacy.

A second system is the ZenCash (now - Horizen) multi-stakeholder governance model. By design, it adopts a flexible multi-stakeholder governance model [?]. The core idea is to remove centralisation which entrusts enormous powers with a minority. Participation is voluntary and decision-making powers cuts across all categories of stakeholders proportional to their resources(stake).

Initially, the Horizen (ZenCash) system has a Core Team (inclusive of founders of Zen) and a DAO (consisting of industry leaders) that controls $3.5\%$ of block mining rewards and $5\%$ of rewards respectively. The plan is to evolve, develop and adopt a hybrid voting mechanism that enables all stakeholders to influence decisions and resource allocations on the blockchain. This evolution would result in a system of DAOs, with competing DAOs responsible for working on different problems. Collectively, the DAOs will be responsible for activities (building, maintaining, improving software, legal, marketing, and advertising) that will ensure the long-term sustainability of Zen.

Community members / stakeholders are allowed to participate in the development of Zen via project proposals which are funded by the DAOs through the $5\%$ block mining reward allocation they receive. We remark that proposals are only to be funded subject to successful voting. Although, at launch, only one DAO "staffed with respected professionals" exists. The staff strength of each DAO is between $3-5$ members and could potentially be increased to any number. A dispute resolution mechanism is to be provided for solving issues among DAO members. Delegative voting is not supported and the system uses fixed amount of voting tokens.

Liquid democracy (also known as delegative democracy [?]) as an hybrid of direct democracy and representative democracy provides the benefits of both system (whilst doing away with their drawbacks) by enabling organisations to take advantage of the experts in a voting process and also gives every member the opportunity to vote [?], [?]. Although the advantages of liquid democracy has been widely discussed in the literature [?], [?], [?], [?], [?], there are few provably secure construction of liquid democracy voting.

Most real-world implementations of liquid democracy only focus on the functionality aspect of their schemes. For instance, Google Vote [?] is an internal Google experiment on liquid democracy over the social media, Google+, which does not consider voter privacy. Similarly, systems such as proxyfor.me [?], LiquidFeedback [?], Adhocracy [?], GetOpinionated, [?] also offer poor privacy guarantees. It is worth mentioning that Sovereign [?] is a blockchain-based voting protocol for liquid democracy; therefore, its privacy is inherited from the underlying blockchain, which provides pseudonymity-based privacy. Wasa2il [?] is able to achieve end-to-end verifiability because this foils privacy. The best known liquid democracy and proxy democracy voting schemes are nVotes [?] and Statement Voting [?], [?]. However, those systems require mix-nets as their underlying primitive. This makes them less compatible to the blockchain setting due to the heavy work load of the mixing servers.

There are a few blockchain based e-voting schemes in the literature, but most of them, e.g., Agora [?], only use the blockchain as a realization of the bulletin board. The actual e-voting schemes are not integrated with the blockchain. [?] is a proposed blockchain-based voting solution that heavily relies on an external "trusted third party" between users and the election authority/authentication authority, in order to ensure anonymity/privacy of voters. Each candidate is voted for by having transactions sent to them. Nonetheless, privacy or anonymity of voters can be broken by collusion between the authentication organisation and the trusted third party. [?] proposes an end-to-end voting system based on Bitcoin that utilises a Kerberos-based protocol to achieve voter identity anonymisation. Voting takes place via sending of tokens from voters to address (public key) of candidates. However, voting is not private and other voters can be influenced by the trend or likelihood of the overall results (before voting is concluded). Furthermore, the scheme is susceptible to coercion.

Our work differs from these earlier works because it not only supports liquid democracy whilst preserving privacy of the voters and delegates, it is also practical in the sense that it considers real-life concerns (e.g., monthly duration of treasury epoch) associated with a treasury system for blockchains.

## IX. Conclusion and Future Work

In this work, we initiated the study of blockchain treasury systems for the community to collaboratively collect and distribute funds in a decentralised manner. We note that the voting scheme used in the treasury system can be further improved with game-theoretic approaches to enable better collaborative decision making. The proposed system can also be extended to serve blockchain self-governance. Our treasury system is planned for practical deployment in cryptocurrencies in 2019. In particular, the treasury model is in the roadmap of Cardano [**?**], to be a part of the Voltaire release [**?**]. Horizen (formerly ZenCash) also implements DAO Treasury Protocol-level Voting System [**?**] based on our scheme.


## Acknowledgment

This work is partially supported by EPSRC grant EP/P034578/1, PETRAS PRF, and IOHK Ltd. We thank Dmytro Kaidalov and Andrii Nastenko from IOHK for the prototype implementation and benchmarks. Bingsheng Zhang is the corresponding author.