A Treasury System for Cryptocurrencies: Enabling Better Collaborative Intelligence *

Bingsheng Zhang[1], Roman Oliynykov[2], and Hamed Balogun[3]

[1] Lancaster University, UK
b.zhang2@lancaster.ac.uk
[2] Input Output Hong Kong Ltd.
roman.oliynykov@iohk.io
[3] 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 the 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.

* This work is supported by IOHK Ltd

Table of Contents

1 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. Collectively, the net market capitalisation of top cryptocurrencies exceeds 400 billion USD.

On the one hand, one of the key features 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.

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, all the aforementioned development funding approaches have risks of centralization in terms of the decision-making on the development steering. Only a few people[4] (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 the 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 [1] 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 very limited. iii) The voting rule and the decision to allow only masternodes to vote in the election makes it "unfairly" difficult for proposals that do not have the support of the founder and core team to succeed because a considerably large amount (about 33%) of masternodes are owned/controlled by the founder and/or core team. This is perfectly captured in a scenario where a proposal despite receiving a decent amount of "YES" votes from other masternodes may not get funding because the core team (33% of the masternodes) voted "NO" against it[5].

[4] For instance, only 4 committee members (i.e. Alex Biryukov, Eran Tromer, Gibson Ashpool, and Zaki Manian) participated in the 2017 Q4 Zcash Grant review process.

[5] In the Dash governance system, a proposal must get at least "+10%" votes in terms of "YES" votes − "NO" votes w.r.t. to all the votes to get shortlisted.

Meanwhile, the concept of cryptographic sortition is proposed by Micali [2]. We can use the idea to randomly sample a small set of users (e.g. 1000 users) with the probability of selection proportional to their corresponding stake. The selected set of users will vote on what projects to be funded. While this is a scalable voting solution, it is not ideal for a treasury system decision-making. This is due to the fact that treasury projects are usually very technical, and normal users may not have solid relevant background (or training) to make wise decisions.

In this work, we propose to use a different approach – *liquid democracy* – to achieve better collaborative intelligence. Liquid democracy (also known as delegative democracy [3]) 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 reknowned 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. Proper selection of the voting scheme allows maximizing the number of voters satisfied by the voting results as well as minimizing voters' effort. There are many voting schemes in the literature. Hereby, we briefly examine two plausible candidates i) *preferential* or *ranked voting* and ii) *approval voting*.

Preferential or ranked voting describes certain voting schemes in which voters rank options (or election candidates) in a hierarchy on the ordinal scale. Its variants include Instant-runoff voting (IRV) [4], Borda count [5], Single transferable vote [6], Schulze method [7], an optimal single-winner preferential voting system (the GT system) based on optimal mixed strategies computation [8], etc. However, as shown in [9], preferential voting has several defects. For instance, adding an outsider to the election candidate list may change results on voting favourites, etc. Besides, strategic behavior of voters may lead them to stepping down their direct preferences (e.g., supporting less preferable proposal because it has higher chances of winning, compared to their most preferred proposal, so as to not have both rejected). This type of strategic behavior hugely affects consensus, maybe positively, however, it may not truly reflect the preferred choices of individuals. Consequently, consensus evaluation for the proposals may give an illusion of a high-level consensus without actual consensus building processes (such as discussions and member interactions) taking place. Furthermore, a disadvantage of using preferential voting is that we may have a deadlock (a.k.a. voting paradox or Condorcet paradox), wherein combined preferences of all individual voters may be cyclic despite individual preferences not being cyclic. Resolving this issue to effectively reflect the sincere voting preference of all participant is very problematic. Moreover, practical application of this voting rule in a treasury system may be too complex for the voters/experts due to the demanding workload on ranking tens or even hundreds of proposals.

Approval voting is an alternative voting method that allows the voters to approve any number of proposals [10]. Winner(s) are chosen by the largest number of supporting ballots. Approval voting is especially suitable for multi-winner elections. It has a number of advantages [11]: simple, quick and easy-to-understand voting process, better expression of true voter intent in his/her ballot, etc.

An extension of approval voting is a "Yes-No-Abstain" type of voting scheme, where the voters express "Yes-No-Abstain" opinion for each proposal. This scheme is used in Dash Governance System [12], The DAO [13], The Fermat Project [14] and other solutions for cryptocurrencies. Recent theoretical analysis of this election rule with variable number of winners, called *Fuzzy threshold voting* [15], 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. In supplementary material A.8 and A.9, we provide the necessary consensus background and analyse consensus level of this voting scheme with examples.

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 [16] via an ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k}$. 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, of course, in that case we don't guarantee protocol termination.

– We propose an efficient design of the treasury system. The system collects fundings 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 the 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. The winning proposals will then be funded.
  Any stakeholder in the community can participate in the treasury voting, and their voting power are 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 [17] 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 [18] 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 committe 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 and deposit paybacks, penalties for faulty actors. All implemented protocols are fully decentralized and resilient up to 50% of malicious or faulty participants. During verification we launched a testnet that consisted of 12 full nodes successfully operating tens of treasury periods with different parameters.

2 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-ℓ 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.

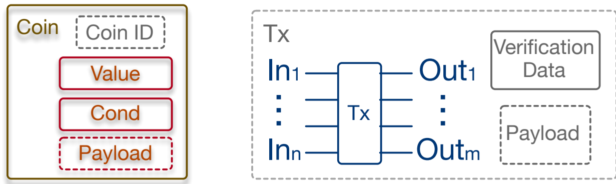

Fig. 1: Coin and transaction structure.

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. 1, 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 the *address*. Conventionally, an address is merely a public key, 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 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.

∘ *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. 1, the transaction has a *Verification data* field that contains the 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 $\mathsf{Payload}$ 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 $\mathsf{Payload}$ field. Note that the verification data is not explicitly described for simplicity.

3 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 welcome to participate. Let k, ℓ, 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 a 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 their stake to participate.
- The experts $\mathcal{E} := \{\mathsf{E}_1, \ldots, \mathsf{E}_m\}$ are a special type of voters that have specialists knowledge and expertise in some field.

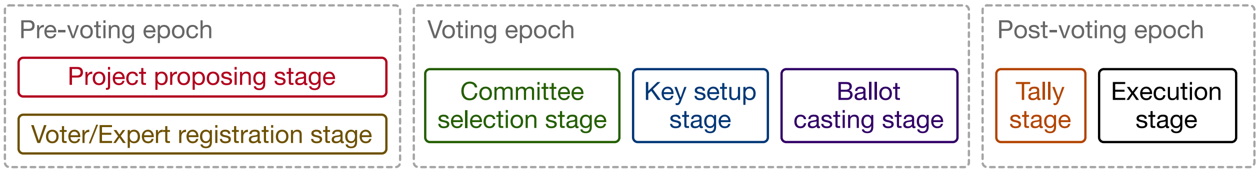

Fig. 2: Treasury system epochs.

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 the stake and the ownership of the actual coin; namely, the stake of the 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 everyone can learn "who owns" the stake of the coin by checking the S-Attr of the coin.

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 2, 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 the 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 in the following sections.) 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 committees will be randomly selected from the registered voters who are willing to be considered for the committee selection. The probability of being selected is proportional to their locked stake. After the voting committee 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, the 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, voting committee members are rewarded (or punished) accordingly. Those 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 be sourced from decentralised means. That is, the 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 it is impossible for all potential funding sources to meet these criteria, a clever combination of some of these potential 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 the block proposer, 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 treasury funds.
– *Minting new coins:* Coin minting is perhaps the most sustainable funding source among all the others. At the beginning of each treasury period, certain amount of coins are created to fund the projects. However, minting may cause inflation in terms of the fiat market value of the underlying cryptocurrency or blockchain platform.
– *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 funding available.

Project proposal. To ensure input independency and eliminate the 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 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 $\mathsf{Payload}$ field, $\textsc{Project}$ is a tag that indicates it is a special project proposal transaction; TID is the treasury ID that is used to uniquely identify a treasury period; $\mathsf{P\text{-}Enc}$ is the encrypted project proposal, and Addr is the return address for the project owner to receive money if the project is 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 forms 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 TCoin is a special output coin whose spending condition is defined in Subsection “supplying the treasury”, below. In the $\mathsf{Payload}$ field, $\textsc{Voter-Reg}$ is a tag that indicates it is a special voter registration transaction; TID is the treasury ID that be 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 $\{\mathsf{S}_i\}_{i=1}^{\ell}$, vk is a freshly generated signature key; and Addr is the return address for the voter to receive treasury reward. The voter's ID is defined as the hash of vk, denoted as $\mathsf{V}_i := \mathsf{hash}(\mathsf{vk})$.

Similarly, 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* in forms of

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

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

The expert's ID is defined as the hash of 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 committee selection epoch, the voting committee of the previous treasury epoch jointly reveal the committed seed, denoted as seed. See supplementary material A.5 for details.

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 vk_i, i.e. st_i is the total stake amount claimed by vk_i. Once seed is announced, any registered voter, who have an address vk_i with claimed stake 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 sk'_i is the corresponding signing key for vk_i, and T is a pre-defined threshold. When the in-equation holds, he/she can submit a special *registration transaction* in forms of

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

where $\{\mathsf{In}_i\}_{i=1}^{n}$ are the input coins, and 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 $\mathsf{Payload}$ field, VC-REG is a tag that indicates it is a special voting committee registration transaction; TID is the treasury ID that be 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 vk_i; and 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 [19] 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 are selected among all λ' committee members. Since $\lambda' = \omega(\log \lambda)$, by Chernoff bound, we have

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

for $\delta = 2\varepsilon/(1-2\varepsilon)$.

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, Cond, specifies that the coin can only be spent according to the corresponding treasury decision. As will be mentioned in details later, 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 λ' 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 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 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. 4, 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 ζ_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 st_i is the amount of the stake claimed by 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 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 joint generate and commit a random seed for the next treasury period. The protocol is depicted in Sec. 4, below.

The first block after treasury period will include all the necessary transactions for treasury funding executions. Those transactions will be signed by all the voting committee members.

4 The proposed voting scheme

In this section, we will describe our decentralized voting schemes that support vote delegation in the UC framework. We first provide the security model in the following.

4.1 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 $\mathcal{F}_{\text{VOTE}}$ during the execution. The ideal functionality $\mathcal{F}_{\text{VOTE}}$ 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 $\mathcal{F}_{\text{VOTE}}$ is depicted in Fig. 3, 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 $\mathcal{F}^{t,k}_{\text{VOTE}}$. The voting will not start until all the committees have participated the preparation phase.

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

The functionality $\mathcal{F}_{\text{VOTE}}^{t,k}$ 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 DelCal (described in Fig. 4) and a tally algorithm TallyAlg (described in Fig. 5) and variables ϕ_1, ϕ_2, τ, 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 ϕ_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 ϕ_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 $\mathcal{F}_{\text{VOTE}}^{t,k}$

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 $\mathcal{F}_{\textsc{Vote}}^{t,k}$. 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 α_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, $\mathcal{F}_{\textsc{Vote}}^{t,k}$ 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 $\mathcal{F}_{\textsc{Vote}}^{t,k}$ to calculate and reveal the delegations received by each expert. After that, they then send $(\textsc{Tally}, sid)$ to the ideal functionality $\mathcal{F}_{\textsc{Vote}}^{t,k}$ to open the tally. Once all the committees have opened the tally, any party can read the tally by sending $(\textsc{ReadTally}, sid)$ to $\mathcal{F}_{\textsc{Vote}}^{t,k}$. Note that due to the natural 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 $\mathsf{TallyAlg}$ is described in Fig. 5.

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 $\mathcal{F}_{\textsc{Ledger}}$ to model the underlying blockchain infrastructure that the treasury system is built on. (cf. supplementary material A.2) We then use the threshold homomorphic encryption functionality $\mathcal{F}_{\text{THVE}}^{t,k}$ to abstract the underlying public key crypto system. (cf. supplementary material A.3) Finally, a global clock functionality $\mathcal{G}_{\textsc{Clock}}$ is adopted to model the synchronised network environment. (cf. supplementary material A.4) Let $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ denote the output of the environment $\mathcal{Z}$ when interacting with parties running the protocol Π and real-world adversary $\mathcal{A}$. Let $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote output of $\mathcal{Z}$ when running protocol ϕ interacting with the ideal functionality $\mathcal{F}$ and the ideal adversary $\mathcal{S}$.

Algorithm DelCal

Input: a set of the expert labels $\mathcal{E}$, and a set of ballots ϕ_2
Output: the delegation result δ

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

Definition 1. *We say that a protocol Π 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* $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

4.2 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 ℓ-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 $\mathbf{e}_i^{(\ell)} = (e_{i,1}^{(\ell)}, \ldots, e_{i,\ell}^{(\ell)})$.

The tally algorithm $\mathsf{TallyAlg}$

Input: a set of the voters $\mathcal{V}$, a set of the experts $\mathcal{E}$, two sets of ballots ϕ_1, ϕ_2 and the delegation δ.
Output: the tally result τ

Init:

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

Tally Computation:

- For each ballot $B \in \phi_2$: parse B in form of $(\mathsf{V}_j, \textsc{Cast}, v_j, \alpha_j)$; 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 $(\mathsf{E}_i, \textsc{Vote}, b_i)$ 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

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 delegate his voting power to the expert E_i. When $y \in \{\textsc{Yes}, \textsc{No}, \textsc{Abstain}\}$, it means that the voter directly vote to 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.

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 occurred 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 $\mathsf{V}_i \in \mathcal{V}$ and an expert $\mathsf{E}_j \in \mathcal{E}$ as $\{\mathsf{In}_i^{(\mathsf{V}_i)}\}_{i=1}^{\ell_1}, \{\mathsf{Out}_j^{(\mathsf{V}_i)}\}_{j=1}^{\ell_2}$ and $\{\mathsf{In}_i^{(\mathsf{E}_j)}\}_{i=1}^{\ell_1}, \{\mathsf{Out}_j^{(\mathsf{E}_j)}\}_{j=1}^{\ell_2}$, respectively.

As depicted in Fig. 6, the treasury voting protocol consists of preparation phase, voting/delegation phase, and tally phase. In the preparation phase, the committee members $\mathsf{C}_j \in \mathcal{C}$ jointly generate the voting public key by sending command $(\textsc{KeyGen}, \mathsf{sid})$ to the functionality $\mathcal{F}_{\text{THVE}}^{t,k}$. In the voting/delegation phase, the voter $\mathsf{V}_i \in \mathcal{V}$ can either delegate his voting power to an expert, or vote directly. The ballot will be encoded as a unit vector $\mathbf{e}^{(m+3)}$. The voter then encrypts the vector by sending $(\textsc{Encrypt}, \mathsf{sid}, \mathbf{e}^{(m+3)})$ to $\mathcal{F}_{\text{THVE}}^{t,k}$, and he receives $(\textsc{Ciphertext}, \mathsf{sid}, \mathbf{u}^{(m+3)})$ from $\mathcal{F}_{\text{THVE}}^{t,k}$. After that, the voter V_i posts $\mathbf{u}^{(m+3)}$ together with the claimed stake α_i to $\mathcal{F}_{\textsc{Ledger}}$ using the macro $\mathsf{Send\text{-}Msg}$ described in Fig. 7. Similarly, the expert $\mathsf{E}_j \in \mathcal{E}$ can express his opinion $v_j \in \{\textsc{Yes}, \textsc{No}, \textsc{Abstain}\}$ by encoding it to a unit vector $\mathbf{e}^3$ and encrypting it as $\mathbf{c}^3$. The expert E_j then posts $\mathbf{c}^3$ to $\mathcal{F}_{\textsc{Ledger}}$ using the macro $\mathsf{Send\text{-}Msg}$. In the tally phase, the committee member $\mathsf{C}_t \in \mathcal{C}$ first fetches all the voting transcripts from the $\mathcal{F}_{\textsc{Ledger}}$. The voter's unit vector are weighted according his stake α_i. All the weighted (encrypted) vectors are then entry-wise added together using additively homomorphic property. The committee first jointly decrypts the delegation part of the resulting unit vector to calculate the delegation. After that, the experts' opinions are weighted according

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

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

Preparation phase:

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

Voting/Delegation phase:

- Upon receiving $(\text{VOTE}, \mathsf{sid}, v_j)$ from the environment $\mathcal{Z}$, the expert E_j, $j \in [m]$ does the following:
  - Send $(\text{READPK}, \mathsf{sid})$ to $\mathcal{F}_{\text{THVE}}^{t,k}$, and receive $(\text{PUBLICKEY}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{F}_{\text{THVE}}^{t,k}$.
  - Set the unit vector $\mathbf{e}^{(3)} \leftarrow \mathsf{encode}^{\mathsf{E}}(v_j)$. Send $(\text{ENCRYPT}, \mathsf{sid}, \mathbf{e}^{(3)}, \eta)$ to $\mathcal{F}_{\text{THVE}}^{t,k}$ and receive $(\text{CIPHERTEXT}, \mathsf{sid}, \mathbf{c}^{(3)})$ from $\mathcal{F}_{\text{THVE}}^{t,k}$.
  - Execute macro $\mathsf{Send\text{-}Msg}\Big(\mathbf{c_j}^{(3)}, \{\mathsf{In}_i^{(\mathsf{E}_j)}\}_{i=1}^{\ell_1}, \{\mathsf{Out}_j^{(\mathsf{E}_j)}\}_{j=1}^{\ell_2}\Big)$. (Cf. Fig. 7)
- Upon receiving $(\text{CAST}, \mathsf{sid}, v_i, \alpha_j)$ from the environment $\mathcal{Z}$, the voter V_i, $j \in [n]$ does the following:
  - Send $(\text{READPK}, \mathsf{sid})$ to $\mathcal{F}_{\text{THVE}}^{t,k}$, and receive $(\text{PUBLICKEY}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{F}_{\text{THVE}}^{t,k}$.
  - Set the unit vector $\mathbf{e}^{(m+3)} \leftarrow \mathsf{encode}^{\mathsf{V}}(v_j)$. Send $(\text{ENCRYPT}, \mathsf{sid}, \mathbf{e}^{(m+3)})$ to $\mathcal{F}_{\text{THVE}}^{t,k}$ and receive $(\text{CIPHERTEXT}, \mathsf{sid}, \mathbf{u}^{(m+3)})$ from $\mathcal{F}_{\text{THVE}}^{t,k}$.
  - Execute macro $\mathsf{Send\text{-}Msg}\Big((\mathbf{u_j}^{(m+3)}, \alpha_j), \{\mathsf{In}_i^{(\mathsf{V}_i)}\}_{i=1}^{\ell_1}, \{\mathsf{Out}_j^{(\mathsf{V}_i)}\}_{j=1}^{\ell_2}\Big)$. (Cf. Fig. 7)

Tally phase:

- Upon receiving $(\text{DELCAL}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the committee C_t, $t \in [k]$ does:
  - Execute macro Read-Msg and obtain data.
  - Fetch the ballots $\{\mathbf{c_i}^{(3)}\}_{i \in [m]}$ and $\{(\mathbf{u_j}^{(m+3)}, \alpha_j)\}_{j \in [n]}$ from data.
  - For $i \in [m]$, send $(\text{CHECK}, \mathsf{sid}, \mathbf{c_i}^{(3)})$ to $\mathcal{F}_{\text{THVE}}^{t,k}$; for $j \in [n]$, send $(\text{CHECK}, \mathsf{sid}, \mathbf{u_j}^{(m+3)})$ to $\mathcal{F}_{\text{THVE}}^{t,k}$; Remove the ballots if the $\mathcal{F}_{\text{THVE}}^{t,k}$ response is not valid.
  - For $j \in [n]$, if a valid $\mathbf{u_j}^{(m+3)}$ is posted, parse $(\mathbf{a_j}^{(m)}, \mathbf{b_j}^{(3)}) := \mathbf{u_j}^{(m+3)}$;
  - For $j \in [n]$, $\ell \in [0, m-1]$, send $(\text{SCALE}, \mathsf{sid}, a_{j,\ell}, \alpha_j)$ to $\mathcal{F}_{\text{THVE}}^{t,k}$ and receive $(\text{SCALE}, \mathsf{sid}, z_{i,\ell})$ from $\mathcal{F}_{\text{THVE}}^{t,k}$.
  - For $i \in [0, m-1]$, send $(\text{ADD}, \mathsf{sid}, (z_{1,i}, \ldots, z_{n,i}))$ to $\mathcal{F}_{\text{THVE}}^{t,k}$ and receive $(\text{SUM}, \mathsf{sid}, s_i)$ from $\mathcal{F}_{\text{THVE}}^{t,k}$;
  - For $i \in [0, m-1]$, send $(\text{DECRYPT}, \mathsf{sid}, s_i)$ to $\mathcal{F}_{\text{THVE}}^{t,k}$.
- Upon receiving $(\text{TALLY}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the committee C_t, $t \in [k]$ does:
  - Send $(\text{READDEC}, \mathsf{sid})$ to $\mathcal{F}_{\text{THVE}}^{t,k}$, and receive $(\text{PLAINTEXT}, \mathsf{sid}, \mathsf{plaintext})$ from $\mathcal{F}_{\text{THVE}}^{t,k}$.
  - Fetch $\{(s_i, w_i)\}_{i \in [m]}$ from plaintext.
  - For $i \in [0, m-1]$, $\ell \in [0, 2]$, send $(\text{SCALE}, \mathsf{sid}, c_{i,\ell}, w_i)$ to $\mathcal{F}_{\text{THVE}}^{t,k}$ and receive $(\text{SCALE}, \mathsf{sid}, d_{i,\ell})$ from $\mathcal{F}_{\text{THVE}}^{t,k}$.
  - For $\ell \in [0, 2]$, send $(\text{ADD}, \mathsf{sid}, (d_{0,\ell}, \ldots, d_{m-1,\ell}, b_{1,\ell}, \ldots, b_{n,\ell}))$ to $\mathcal{F}_{\text{THVE}}^{t,k}$ and receive $(\text{SUM}, \mathsf{sid}, x_\ell)$ from $\mathcal{F}_{\text{THVE}}^{t,k}$;
  - For $\ell \in [0, 2]$, send $(\text{DECRYPT}, \mathsf{sid}, x_\ell)$ to $\mathcal{F}_{\text{THVE}}^{t,k}$.
- Upon receiving $(\text{READTALLY}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the party P does the following:
  - Send $(\text{READDEC}, \mathsf{sid})$ to $\mathcal{F}_{\text{THVE}}^{t,k}$, and receive $(\text{PLAINTEXT}, \mathsf{sid}, \mathsf{plaintext})$ from $\mathcal{F}_{\text{THVE}}^{t,k}$.
  - Fetch $\{(x_\ell, y_i)\}_{i \in [0,2]}$ from plaintext, and return $(\text{READTALLYRETURN}, \mathsf{sid}, (y_0, y_1, y_2))$ to the environment $\mathcal{Z}$.

Fig. 6: The voting protocol $\Pi_{\text{VOTE}}^{t,k,m,n}$ in $\{\mathcal{F}_{\text{LEDGER}}, \mathcal{F}_{\text{THVE}}^{t,k}\}$-hybrid model

to their delegations, and the weighted opinions are aggregated. Finally the committees $\mathcal{C}$ jointly decrypt the final tally. After that, any party can read the tally result on the $\mathcal{F}_{\mathrm{THVE}}^{t,k}$.

Sending/Reading data to/from $\mathcal{F}_{\mathbf{Ledger}}$. Fig. 7 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 [20], 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 Δ_1 rounds, which is $2\Delta_1$ clock-ticks in [20]. 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 denial-of-service such a transaction. This is modeled by the ExtendPolicy in $\mathcal{F}_{\text{LEDGER}}$, where if a transaction is more than Δ_2 rounds (i.e. $2\Delta_2$ clock-ticks) old, and 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 [21]. Hence, all the honest parties can have a common state of any blocks that have been proposed more than windowsize. Denote Δ_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 (REGISTER, 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 (REGISTER, 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 until the transaction that carries the intended message to be included in the state of the party.

Sending and reading messages

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

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

Macro Read-Msg**:**

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

Fig. 7: Macro for sending and receiving message via $\mathcal{F}_{\text{LEDGER}}$

5 A new unit vector ZK proof

In this section, we propose a new unit vector zero-knowledge proof/argument with logarithmic size communication. Before describing our construction, we refer interested reader to supplementary material A.6 for necessary definitions and primitives.

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 [17] 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 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 [22], 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. 8.

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 $\mathbf{e}_i^{(n)}$

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 [22]. 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 β_ℓ 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.8, 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. 9, 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_ℓ, $\ell \in [\log n]$. Subsequently, it produces B_ℓ, A_ℓ as the first move of the Sigma protocol in Sec. 2.3 of [22] showing I_ℓ 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 the computes and sends the aforementioned $\{D_\ell\}_{\ell=0}^{\log n-1}$. The verification consists of two parts. In the first part, the verifier checks the following equations to ensure that I_ℓ 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.

Unit vector ZK argument

CRS: the commitment key ck
Statement: the public key 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 $D_\ell := \mathsf{Enc}_{\mathsf{pk}}\left(\sum_{j=0}^{n-1}(p_{j,\ell} \cdot y^j); R_\ell\right)$
- $P \to V$: $\{D_\ell\}_{\ell=0}^{\log n-1}$;
- $V \to P$: 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:
  - $(I_\ell)^x \cdot B_\ell = \mathsf{Com}_{\mathsf{ck}}(z_\ell; w_\ell)$
  - $(I_\ell)^{x-z_\ell} \cdot A_\ell = \mathsf{Com}_{\mathsf{ck}}(0; v_\ell)$
- $\prod_{j=0}^{n-1}\left((C_j)^{x^{\log n}} \cdot \mathsf{Enc}_{\mathsf{pk}}(-\prod_{\ell=1}^{\log n} z_{\ell,j_\ell}; 0)\right)^{y^j} \cdot \prod_{\ell=0}^{\log n-1}(D_\ell)^{x^\ell} = \mathsf{Enc}_{\mathsf{pk}}(0; R)$, where $z_{j,1} = z_j$ and $z_{j,0} = x - z_j$.

Fig. 9: Unit vector ZK argument

Theorem 1. *The protocol described in Fig. 9 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* $C_j = \mathsf{Enc}_{\mathsf{pk}}(e_{i,j}; r_j)$, $j \in [0, n-1]$.

Proof. See Supplementary Material A.6.

6 Security

The security of the treasury voting protocol is analysed in the UC framework. We provide Theorem 2 and its proof can be found in supplementary material A.7.

Theorem 2. *Let $k, n, m = \mathrm{poly}(\lambda)$ and $t > k/2$. Protocol $\Pi_{\mathrm{VOTE}}^{t,k,n,m}$ described in Fig. 6 UC-realizes $\mathcal{F}_{\mathrm{VOTE}}^{t,k}$ in the $\{\mathcal{F}_{\mathrm{LEDGER}}, \mathcal{F}_{\mathrm{THVE}}^{t,k}\}$-hybrid world against static corruption.*

7 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 [23] 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 [19] 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 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 expers, 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. 10.

Besides it, there is an estimated amount of data needed to be transmitted over a peer-to-peer network to complete the protocol, in dependence of committee size and malicious members ratio. Results are given in Fig. 11 (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 very small 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 12 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. 13. In particular, for a treasury period with 5000 voters and 50 experts, the overall communication is approximately 20 MB per project.

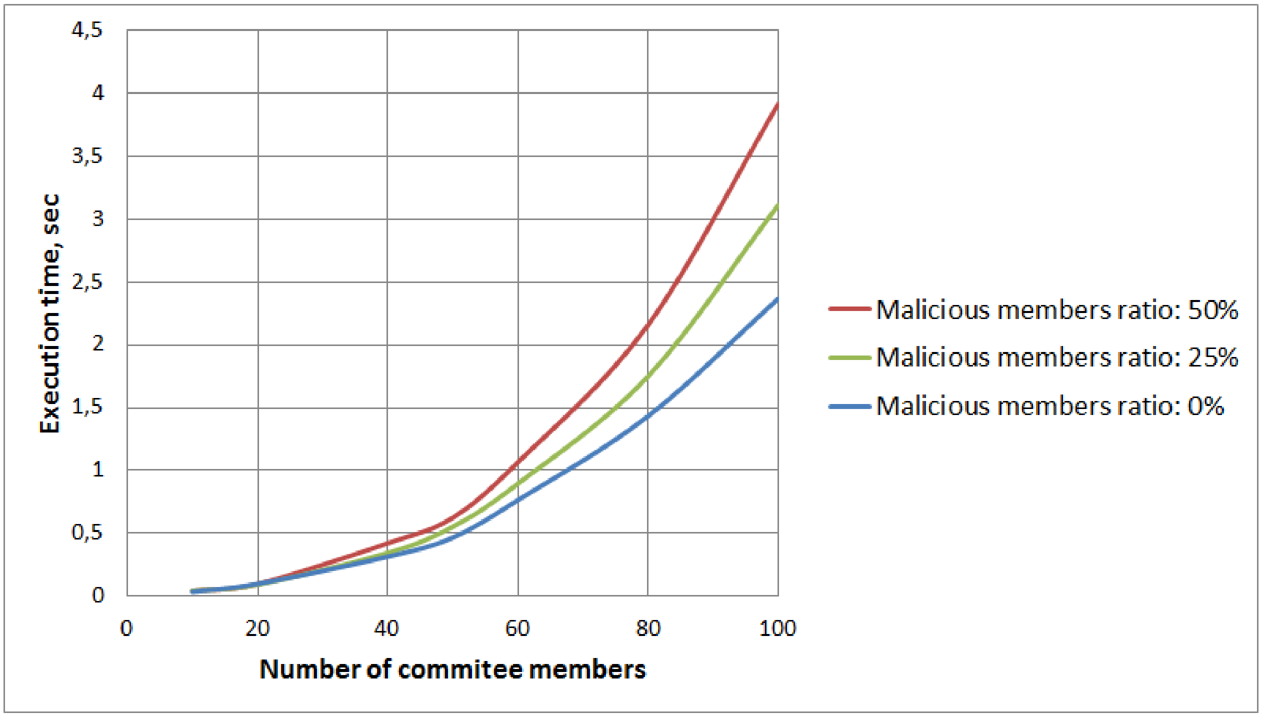

Fig. 10: DKG protocol execution time depending on the number of committee members

8 Related work

The Dash governance system (DGS) [1] 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, however it is affected by a number of security and centralisation issues. For instance, voting on the DGS is not private, thereby leaving nodes susceptible to coercion.

Beyond voting, the Dash Governance System (DGS) [1,24], is the first self-sustenance/funding mechanism in any cryptocurrency or blockchain system. However, the DGS does not support delegative voting and ballot privacy. Amongst other drawbacks, only operators of MasterNodes are allowed to propose projects and vote and about 73% of all funded proposals have been proposed by two members of the DASH community.

A second system is the Zencash multi-stakeholder governance model. By design, ZenCash adopts a flexible multi-stakeholder governance model [25]. The core idea is to remove centralisation which entrusts enormous

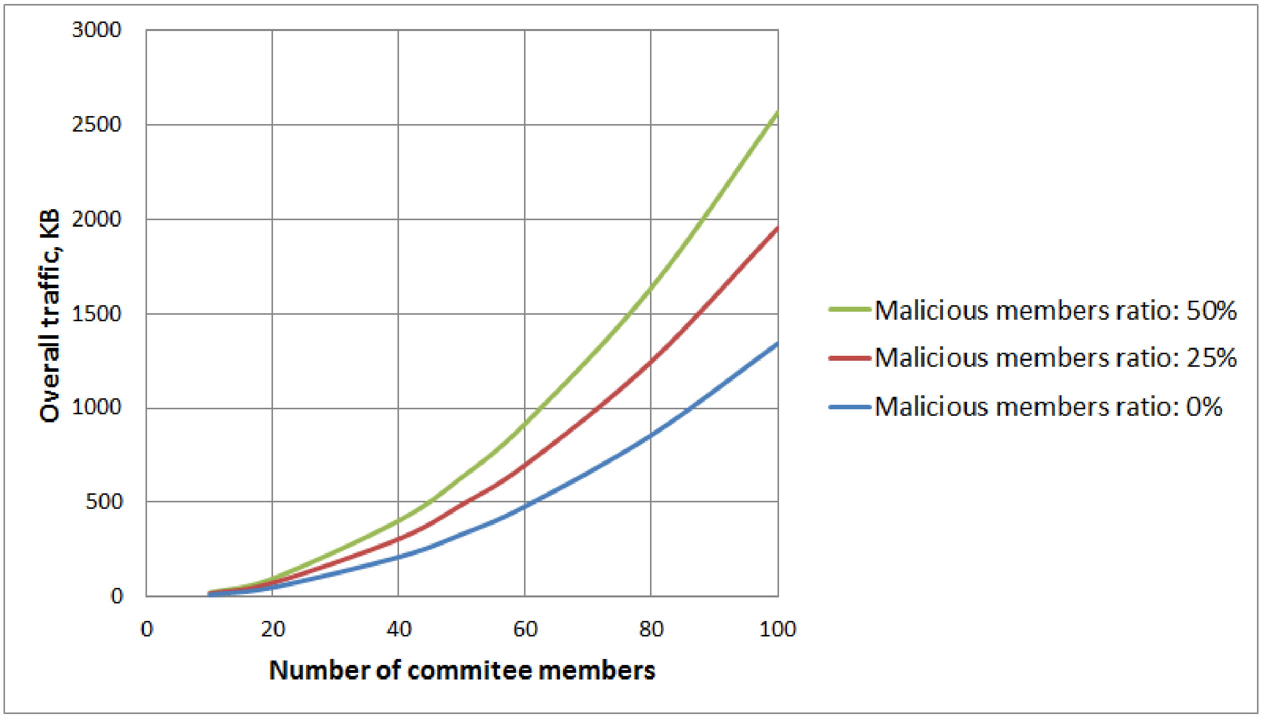

Fig. 11: Total size of the DKG protocol messages to be sent over the peer-to-peer network depending on the number of committee members

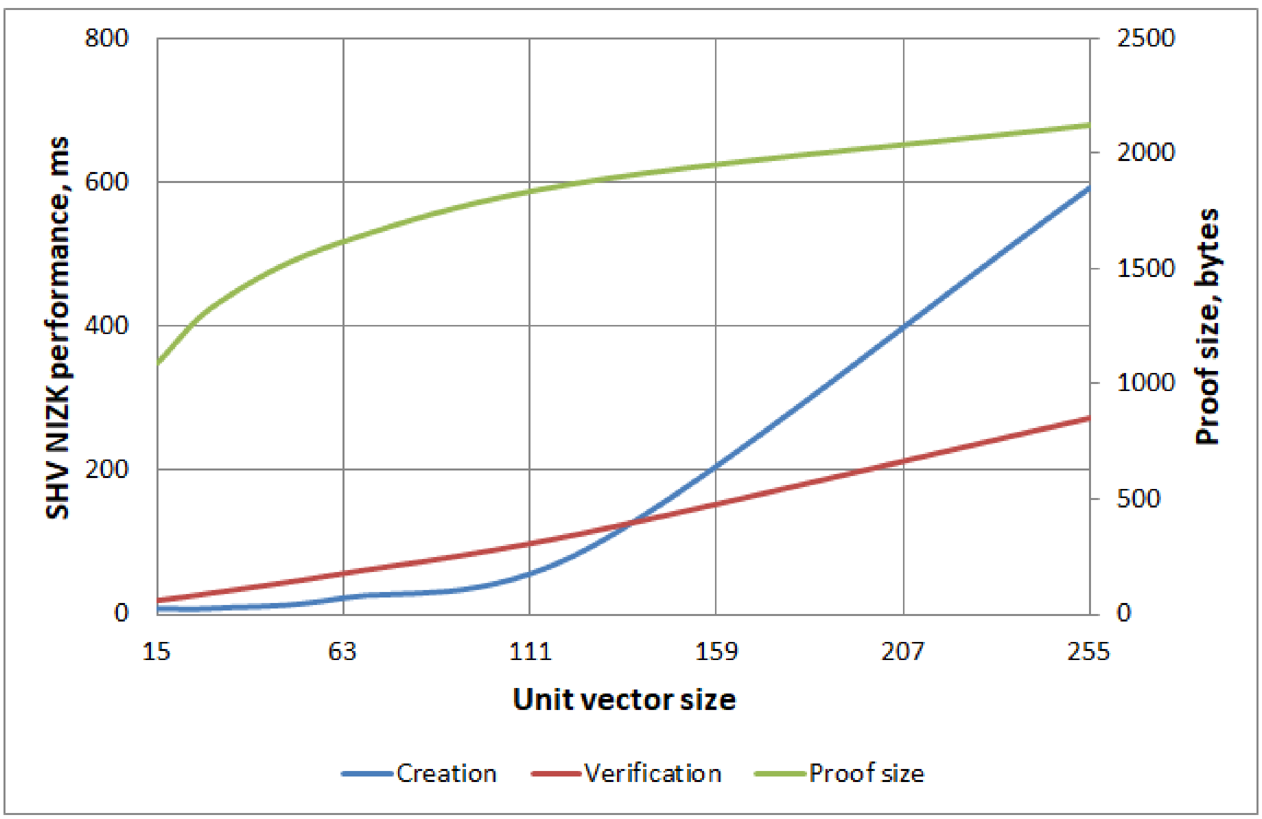

Fig. 12: The prover's running time, verifier's running time and the size of the unit vector ZK proof.

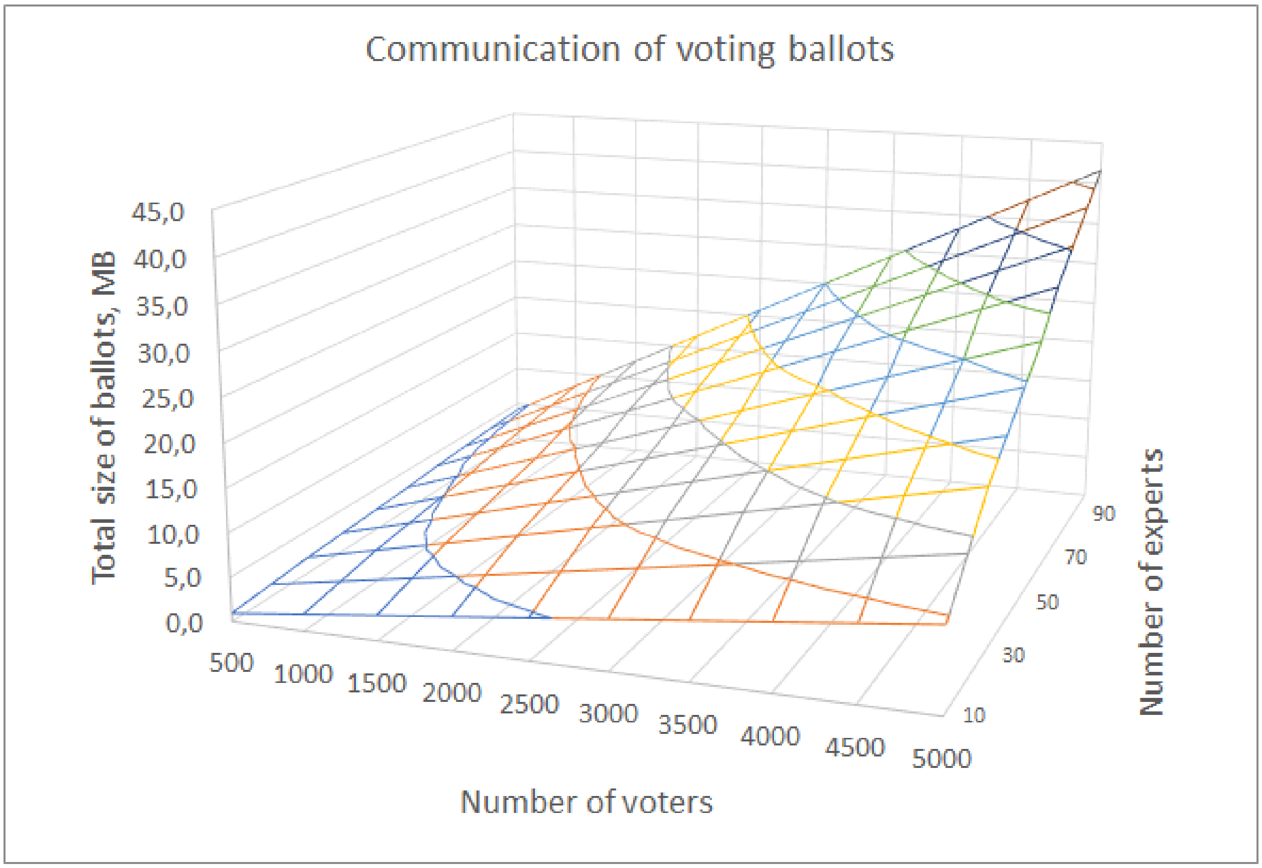

Fig. 13: The overall communication for all the voting ballots during an entire treasury period.

powers with a minority. Participation is voluntary and decision-making powers cuts across all categories of stakeholders proportional to their resources(stake).

Initially, the 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 obviously 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 (or currently), only one DAO "staffed with respected professionals" exists. The staff strength of each DAO is between $3-5$ members and could potentially be increased into any number. A dispute resolution mechanism is to be provided for solving issues between DAO members.

Although Zencash's attempts to enable multi-stakeholder participation in governance is commendable, the system/structure has some obvious drawbacks. Unlike DGS, voting is open to all community members, but is susceptible to Denial of Service attacks. Delegative voting is not supported and the system uses fixed amount of voting tokens and too many "magic numbers". Furthermore, veto power is granted to the core Team comprising only 3 individuals.This can lead to a concept that described as "Tyranny of the powerful/founders".

Liquid democracy (also known as delegative democracy [3]) 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 [30]. Although the advantages of liquid democracy has been widely discussed in the literature [31,32,33,34,35], 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 [36] 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 [37], LiquidFeedback [38], Adhocracy [39], GetOpinionated,[40] also offer poor privacy guarantees. It is worth mentioning that Sovereign [41] is a blockchain-based voting protocol for liquid democracy; therefore, its privacy is inherited from the underlying blockchain, which provides pseudonymity-based privacy. Wasa2il [42] is able to achieve End-to-End verifiability because this foils privacy. The best known liquid democracy and proxy democracy voting schemes are nVotes [43] and Statement Voting [44]. However, those systems require mix-net as their underlying primitive. This makes them less compatible to the blockchain setting due to the heavy work load of the mixing servers.

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 factors in real-life concerns (e.g., monthly duration of treasury epoch) associated with a treasury system for blockchains. In a worse case scenario, our privacy guarantees are equivalent to that obtainable in the Dash Governance System.

9 Acknowledgments

We would like to thank Dmytro Kaidalov and Andrii Nastenko from IOHK for the full prototype implementation of the proxy voting protocol, all its sub-protocols and algorithms, distributed key generation and their performance benchmarks. Many thanks to Tamara Finogina for many enlightening discussions, comments and insights.

References

1. E. Duffield and D. Diaz, "Dash: A privacycentric cryptocurrency."
2. S. Micali, "ALGORAND: the efficient and democratic ledger," *CoRR*, vol. abs/1607.01341, 2016.
3. B. Ford, "Delegative democracy," *Manuscript*, 2002.
4. D. Cary, "Estimating the margin of victory for instant-runoff voting.," in *EVT/WOTE*, 2011.
5. P. Emerson, "The original borda count and partial voting," *Social Choice and Welfare*, pp. 1–6, 2013.
6. N. Tideman, "The single transferable vote," *The Journal of Economic Perspectives*, vol. 9, no. 1, pp. 27–38, 1995.
7. M. Schulze, "A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method," *Social Choice and Welfare*, vol. 36, no. 2, pp. 267–303, 2011.
8. R. L. Rivest and E. Shen, "An optimal single-winner preferential voting system based on game theory," in *Proc. of 3rd International Workshop on Computational Social Choice*, pp. 399–410, 2010.
9. P. C. Fishburn and S. J. Brams, "Paradoxes of preferential voting," *Mathematics Magazine*, vol. 56, no. 4, pp. 207–214, 1983.
10. S. J. Brams and P. C. Fishburn, "Approval voting," *American Political Science Review*, vol. 72, no. 3, pp. 831–847, 1978.
11. "Approval voting." Online; date last accessed: 2017-10-26.
12. E. Duffield and D. Diaz, "Dash: A privacy-centric crypto-currency," 2014.
13. D. Harrison, "Decentralized autonomous organizations," May 16 2016. Online; date last accessed: 2017-10-21.
14. L. F. Molina, "Fermat, the internet of people and the person to person economy," 2016. Online; date last accessed: 2017-10-26.
15. D. Kaidalov, L. Kovalchuk, *et al.*, "A proposal for an ethereum classic treasury system."
16. R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols." Cryptology ePrint Archive, Report 2000/067, 2000. https://eprint.iacr.org/2000/067.
17. D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *CRYPTO 1992 Proceedings*, vol. 740 of *LNCS*, pp. 89–105, Springer, 1993.
18. "Treasury prototype implementation."
19. A. Chepurnoy, T. Duong, L. Fan, and H. Zhou, "Twinscoin: A cryptocurrency via proof-of-work and proof-of-stake," *IACR Cryptology ePrint Archive*, vol. 2017, p. 232, 2017.
20. C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas, "Bitcoin as a transaction ledger: A composable treatment," in *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I* (J. Katz and H. Shacham, eds.), vol. 10401 of *Lecture Notes in Computer Science*, pp. 324–356, Springer, 2017.
21. J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II* (E. Oswald and M. Fischlin, eds.), vol. 9057 of *Lecture Notes in Computer Science*, pp. 281–310, Springer, 2015.
22. J. Groth and M. Kohlweiss, *One-Out-of-Many Proofs: Or How to Leak a Secret and Spend a Coin*, pp. 253–280. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015.
23. "Scorex 2 - the modular blockchain framework."
24. D. Kaidalov, A. Nastenko, *et al.*, "Dash governance system: Analysis and suggestions for improvements."
25. R. Viglione, R. Versluis, and J. Lippencott, "Zen white paper," 2017.
26. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *EUROCRYPT '99*, (Berlin, Heidelberg), pp. 295–310, Springer Berlin Heidelberg, 1999.
27. M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, (New York, NY, USA), pp. 62–73, ACM, 1993.
28. A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO' 86*, (Berlin, Heidelberg), pp. 186–194, Springer Berlin Heidelberg, 1986.
29. J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *J. ACM*, vol. 27, pp. 701–717, Oct. 1980.
30. B. Zhang and H. Zhou, "Brief announcement: Statement voting and liquid democracy," in *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pp. 359–361, 2017.
31. D. Lomax, "Beyond politics, an introduction," January 1, 2003. Online; date last accessed: 2017-10-21.
32. P. Boldi, F. Bonchi, C. Castillo, and S. Vigna, "Voting in social networks," in *Proceedings of the 18th ACM conference on Information and knowledge management*, pp. 777–786, ACM, 2009.
33. M. Nordfors, "Democracy 2.1: How to make a bunch of lazy and selfish people work together," 2003. Online; date last accessed: 2017-10-21.

34. J. Green-Armytage, "Direct voting and proxy voting," *Constitutional Political Economy*, vol. 26, no. 2, pp. 190–220, 2015.
35. J. Ito, "Emergent democracy," *Retrieved November*, vol. 17, p. 2007, 2003.
36. S. Hardt and L. C. Lopes, "Google votes: A liquid democracy experiment on a corporate social network," 2015.
37. Proxy.me, "Voteflow," 2015. Online; date last accessed: 2017-10-21.
38. LiquidFeedback, "LiquidFeedback official website." Online; date last accessed: 2017-10-21.
39. Adhocracy, "Adhocracy official website." Online; date last accessed: 2017-10-21.
40. J. Degrave, "Getopinionated." GitHub repository; date last accessed: 2017-10-21.
41. Democracy Earth, "The social smart contract. an open source white paper.," September 1, 2017. Online; date last accessed: 2017-10-21.
42. S. McCarthy, "Wasa2il," 2016. GitHub repository; date last accessed: 2017-10-21.
43. nVotes, "3 crypto schemes for liquid democracy (iii)," July 19 2017. Online; date last accessed: 2017-10-21.
44. B. Zhang and H.-S. Zhou, "Statement voting." Cryptology ePrint Archive, Report 2017/616, 2017. http://eprint.iacr.org/2017/616.
45. J. E. Innes and D. E. Booher, "Consensus building and complex adaptive systems," *Journal of the American Planning Association*, vol. 65, no. 4, pp. 412–423, 1999.
46. J. Trimbur, "Consensus and difference in collaborative learning," *College English*, vol. 51, no. 6, p. 602, 1989.
47. R. A. Irvin and J. Stansbury, "Citizen participation in decision making: Is it worth the effort?," *Public Administration Review*, vol. 64, pp. 55–65, January 2004.
48. E. Herrera-Viedma, F. Herrera, and F. Chiclana, "A consensus model for multiperson decision making with different preference structures," *Trans. Sys. Man Cyber. Part A*, vol. 32, pp. 394–402, May 2002.
49. S. Boroushaki and J. Malczewski, "Measuring consensus for collaborative decision-making: A gis-based approach," *Computers, Environment and Urban Systems*, vol. 34, no. 4, pp. 322 – 332, 2010. Geospatial Cyberinfrastructure.
50. R. R. YAGER and D. P. FILEV, "Parameterized and-uke and or-like owa operators," *International Journal of General Systems*, vol. 22, no. 3, pp. 297–316, 1994.

A Supplementary material

A.1 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 within) 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.

A.2 The blockchain ideal functionality

We adopt the state-of-the-art blockchain ideal functionality proposed by Badertscher et al., [20]. For completeness, we recap the functionality here. As shown, in Fig. 14, the functionality maintains the set of registered parties $\mathbb{P}$, the (sub-)set of honest parties $\mathcal{H} \subseteq \mathbb{P}$, and the (sub-set) of de-synchronized honest parties $\mathbb{P}_{DS} \subset \mathcal{H}$. The set $\mathbb{P}, \mathbb{P}_{DS}, \mathcal{H}$ are all initially set to $\emptyset$. When a new honest party is registered, it is added to all $\mathbb{P}_{DS}$ (hence also to $\mathcal{H}$ and $\mathbb{P}$ and the current time of registration is also recorded; similarly, when a party is deregistered, it is removed from both $\mathbb{P}$ and $\mathbb{P}_{DS}$. For each party $p \in \mathbb{P}$, the functionality maintains a pointer pt_i (initially set to 1) and a current state view $\mathsf{state}_\mathsf{i} := \epsilon$ (initially set to empty). The functionality also keeps track of the timed honest-input sequence in a vector $\mathsf{I}_\mathsf{H}^\mathsf{T}$ (initially $\mathsf{I}_\mathsf{H}^\mathsf{T} := \epsilon$)

Functionality $\mathcal{F}_{\text{LEDGER}}$

It is parametrized by four algorithms Validate, ExtendPolicy, Blockify, and predict-time, along with two parameters: windowSize, Delay $\in \mathbb{N}$. The functionality manages variables state, NxtBC, buffer, τ_L and τ_{state}.
Initially, state $:= \tau_{\textbf{state}} :=$ NxtBC $:= \epsilon$, buffer $:= \emptyset, \tau_{\text{L}} = 1$.
The functionality maintains the set of registered parties $\mathbb{P}$, the (sub-)set of honest parties $\mathcal{H} \subseteq \mathbb{P}$, and the (sub-set) of de-synchronized honest parties $\mathbb{P}_{DS} \subset \mathcal{H}$. The set $\mathbb{P}, \mathbb{P}_{DS}, \mathcal{H}$ are all initially set to $\emptyset$. When a new honest party is registered, it is added to all $\mathbb{P}_{DS}$ (hence also to $\mathcal{H}$ and $\mathbb{P}$ and the current time of registration is also recorded; similarly, when a party is deregistered, it is removed from both $\mathbb{P}$ and $\mathbb{P}_{DS}$. For each party $p \in \mathbb{P}$, the functionality maintains a pointer pt_i (initially set to 1) and a current state view $\text{state}_\text{i} := \epsilon$ (initially set to empty). The functionality also keeps track of the timed honest-input sequence in a vector $\text{I}_\text{H}^\text{T}$ (initially $\text{I}_\text{H}^\text{T} := \epsilon$)

Upon receiving any input I from any party or from the adversary, send (GetTime, sid) to $\mathcal{G}_{\text{Clock}}$ and upon receiving response (GetTime, sid, τ) set $\tau_L := \tau$ and do the following:

- Let $\hat{\mathbb{P}} \subseteq \mathbb{P}_{DS}$ denote the set of desynchronized honest parties that were registered at time $\tau' \leq \tau_L -$ Delay. Set $\mathbb{P}_{DS} := \mathbb{P}_{DS} \setminus \mathbb{P}$.
- If I was received from an honest party $p \in \mathbb{P}$:
  - Set $\mathbf{I_H^T} := \mathbf{I_H^T} || (I, p, \tau_L)$;
  - Compute $\mathbf{N} = (\mathbf{N_1}, \dots, \mathbf{N_l}) :=$ ExtendPolicy($\mathbf{I_H^T}$, state, NxtBC, buffer, $\tau_{\textbf{state}}$) and if $\mathbf{N} \neq \epsilon$ set state := state||Blockify(($\mathbf{N_1 1}$))|| ..., Blockify(($\mathbf{N_l}$)) and $\tau_{\textbf{state}} := \tau_{\textbf{state}} || \tau_L^l$, where $\tau_L^l = \tau_L ||, \dots \tau_L$
  - For each BTX $\in$ buffer: if (Validate, BTX, state, buffer,) $= 0$ then delete BTX from buffer. Also reset NxtBC $:= \epsilon$
  - If there exists $p_j \in \mathcal{H}$ such that $|\text{state}| - pt_j >$ windowSize or $pt_j < |\text{state}|$, then set $pt_k := |\text{state}|$ for all $p_k \in \mathcal{H} \setminus \mathbb{P}_{DS}$
- Depending on the above input I and its sender's ID, $\mathcal{F}_{\text{LEDGER}}$ executes the corresponding code from the following list:
  - Submitting a transaction: If $I = (\text{Submit}, sid, tx)$ and is received from a party $p \in \mathbb{P}$ or from $\mathcal{A}$ (on behalf of a corrupted party p) do the following
    * Choose a unique transaction ID txid and set BTX $:= (tx, txid, \tau_L, p_i)$
    * if Validate(BTX), state, buffer $= 1$, then buffer := buffer $\cup$ {BTX}.
    * Send (Submit, BTX) to $\mathcal{A}$
  - Reading the state: If $I = (\text{Read}, sid)$ is received from a party $p \in \mathbb{P}$ then set $\text{state}|_{\min\{\text{pt}_\text{i}, |\text{state}|\}}$ and return (Read, sid, state_i) to the requester. If the requester is $\mathcal{A}$ then send (state, buffer, $\mathbf{I_H^T}$) to $\mathcal{A}$
  - Maintaining the ledger state: If $I = (\text{Maintain-Ledger}, sid, minerID)$ is received by an honest party p in $\mathbb{P}$ and (after updating $\mathbf{I_H^T}$ as above) predcit-time($\mathbf{I_H^T}$) $= \hat{\tau} > \tau_L$ then send (Tick, sid) to $\mathcal{G}_{\text{Clock}}$. Else, send I to $\mathcal{A}$.
  - The adversary proposing the next block: If $I = (\text{Next-Block}, \text{hFlag}, (txid_1, \dots, txid_l))$ is sent from the adversary, update NxtBC as follows:
    * Set listOfTxid $\leftarrow \epsilon$
    * For $i = 1, \dots, l$ do: if there exists BTX $:= (x, txid, minerID, \tau_L, p_i) \in$ buffer with ID $txid = txid_i$ then set lisOfTxid := listOfTxid$||txid_i$
    * Finally, set NxtBC := NxtBC$||(hFlag, listOfTxid)$ and output (Next-Block, ok) to $\mathcal{A}$
  - The adversary setting state-slackness: If $I = (\text{Set-Slack}, (p_{i1}, \hat{pt}_{i1}), (p_{il}, \hat{pt}_{il}),)$ with $\{p_{i1}, \dots, p_{il}\} \subseteq \mathcal{H} \setminus \mathbb{P}_{DS}$ is received from the adversary $\mathcal{A}$ do the following:
    * If for all $j \in [l]$: $|\text{state}| - \hat{pt}_{ij} \geq \text{state}_{\text{ij}}|, setpt_i := \hat{pt}_i$ for every $j \in [l]$ and return (Set-Slack, ok) to $\mathcal{A}$.
    * Otherwise set $pt_j :=$ state for all $j \in [l]$
  - The adversary setting the state for desynchronised parties: If $I = (\text{DEesync-State}, (p_{i1}, \text{state}'_{\text{il}}))$ with $\{p_{i1}, \dots, p_{il}\} \subseteq \mathbb{P}_{DS}$ is received from the adversary $\mathcal{A}$, set $\text{state}_{\text{ij}} := \text{state}'_{\text{ij}}$ for each $j \in [l]$ and return (Desync-State, ok) to $\mathcal{A}$.

Fig. 14: Functionality $\mathcal{F}_{\text{LEDGER}}$

A.3 The threshold homomorphic encryption functionality

The ideal functionality is depicted in Fig. 15. It is parameterized by the additively homomorphic public key encryption algorithms $\mathsf{HE} = (\mathsf{KeyGen}^{\mathsf{E}}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Scale}, \mathsf{Dec})$. To generate the key, the key holders $\mathsf{K}_i \in \mathcal{K}$ sends $(\textsc{KeyGen}, \mathsf{sid})$ command to the functionality $\mathcal{F}_{\mathrm{THVE}}^{t,k}$. Upon receiving the key generation requests from at least $t > k/2$ key holders, $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ generates the key pairs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}^{\mathsf{E}}(\mathsf{param})$. Any party can read the public key by sending $(\textsc{ReadPK}, \mathsf{sid})$ command to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$. To encrypt a vector, any party can send $(\textsc{Encrypt}, \mathsf{sid}, \mathbf{m}^{(\ell)} := (m_0, \ldots, m_{\ell-1}), \mathsf{pk}', \eta)$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$, where $\eta = 0$ if the party does not want to prove that $\mathbf{m}^{(\ell)}$ is a unit vector; otherwise, $\eta = 1$, if the party want to show that $\mathbf{m}^{(\ell)}$ is a unit vector. In the latter case, the functionality $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ checks the validity of $\mathbf{m}^{(\ell)}$ and then put the corresponding ciphertext in $\mathsf{unit\text{-}vec}$. Any party can query $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ to check whether $\mathbf{c}^{(\ell)})$ is a unit vector. If the ciphertext vector is in $\mathsf{unit\text{-}vec}$, $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ returns valid; otherwise, it returns $\mathsf{unknown}$. Moreover, the user can run homomorphic operations Add and Scale on the ciphertexts if they were encrypted under pk. The Decryption function allows for the decryption of individual ciphertexts (or revealing of shares) provided a certain threshold of ciphertexts have not earlier been revealed.

The ideal functionality $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ can be efficiently realized from threshold Elgamal encryption in practice. Distributed key generation (DKG) is a fundamental building block of such a protocol. To ensure robustness, the elected voting committee members invoke the distributed key generation protocol to setup the voting public key. Ideally, the protocol termination should be guaranteed when up to $t = \lceil \frac{n}{2} \rceil - 1$ out of n committee members are corrupted. A naive way of achieving threshold distributed key generation is as follows. Each of the voting committee members C_i first generates a public/private key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}^{\mathsf{E}}(\mathsf{param})$. Each C_i then posts pk_i to the blockchain and use $(t+1, n)$-threshold *verifiable secret sharing* (VSS) to share sk_i to all the other committee members. The combined voting public key can then be defined as $\mathsf{pk} := \prod_{i=1}^{n} \mathsf{pk}_i$. However, this approach is problematic in the sense that the adversary can influence the distribution of the final voting public key by letting the corrupted committee members abort selectively. Alternatively, we will adopt the distributed key generation protocol proposed by Gennaro *et al.* [26]. In a nutshell, the protocol lets the committee members C_i first posts a "commitment" of pk_i. After sharing the corresponding sk_i via $(t+1, n)$-threshold VSS, the committee members C_i then reveals pk_i. We will use the blockchain to realise the broadcast channel and peer-to-peer channels.

A.4 The global clock functionality

The global clock functionality $\mathcal{G}_{\mathrm{CLOCK}}$ interacts with all the parties. To handle offline parties, the parties can register and deregister themselves to the functionality $\mathcal{G}_{\mathrm{CLOCK}}$, and the clock will advance if and only if all the registered honest parties have sent $\textsc{Tick}$ command to it.

A.5 Supplementary material for Section 3

To generate and commit a random seed, voting committee members C_ℓ, $\ell \in [k]$ 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 pk. More specifically, each voting committee members C_ℓ, $\ell \in [k]$ 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.

A.6 Supplementary material for Section 5

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 ϕ is a public predicate function.

Threshold homomorphic encryption functionality $\mathcal{F}_{\text{THVE}}^{t,k}$

Ideal Functionality $\mathcal{F}_{\text{THVE}}^{t,k}$

Threshold homomorphic vector encryption. The functionality interacts with a set of key holders $\mathcal{K} := \{\mathsf{K}_1, \ldots, \mathsf{K}_k\}$ and a set of users $\mathcal{U} := \{\mathsf{U}_1, \ldots, \mathsf{U}_n\}$, and the adversary $\mathcal{A}$. Let $\mathcal{K}_{\mathsf{honest}}$ and $\mathcal{K}_{\mathsf{cor}}$ denote the honest and corrupt sets of key holders, respectively. It is parameterized by the system parameters $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$, a set of encryption algorithms $\mathsf{HE} = (\mathsf{KeyGen}^{\mathsf{E}}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Scale}, \mathsf{Dec})$ and variables $\mathsf{ciphertext}$, $\mathsf{plaintext}$, and $\mathsf{unit\text{-}vec}$. Initially, $\mathsf{ciphertext} := \emptyset$, $\mathsf{plaintext} := \emptyset$ and $\mathsf{unit\text{-}vec} := \emptyset$.

Key Generation:

- Upon receiving input $(\textsc{KeyGen}, \mathsf{sid})$ from $\mathsf{K}_i \in \mathcal{K}$, send a notification message $(\textsc{KeyGenNotify}, \mathsf{sid}, \mathsf{K}_i)$ to the adversary $\mathcal{A}$. If it has collected $\textsc{KeyGen}$ request from at least t key holders K_i, then it generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}^{\mathsf{E}}(\mathsf{param})$.
- Upon receiving input $(\textsc{ReadPK}, \mathsf{sid})$ from party $P \in \mathcal{K} \cup \mathcal{U}$, if pk has been recorded, then return $(\textsc{PublicKey}, \mathsf{sid}, \mathsf{pk})$ to P.

Encryption:

- Upon receiving $(\textsc{Encrypt}, \mathsf{sid}, \mathbf{m}^{(\ell)} := (m_0, \ldots, m_{\ell-1}), \mathsf{pk}', \eta)$ from any party $P \in \mathcal{K} \cup \mathcal{U}$:
  - If $\mathsf{pk} \neq \mathsf{pk}'$, then for $i \in [0, \ell-1]$, compute $c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}'}(m_i)$, and send $(\textsc{Ciphertext}, \mathsf{sid}, c_0, \ldots, c_n)$ to party P.
  - Else if $\mathsf{pk} = \mathsf{pk}'$, then for $i \in [0, \ell-1]$: compute $c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$; record pair (c_i, m_i) in $\mathsf{ciphertext}$; init $J_{c_i} := \emptyset$. Send $(\textsc{Ciphertext}, \mathsf{sid}, \mathbf{c}^{(\ell)} := (c_0, \ldots, c_{\ell-1}))$ to party P. If $\eta = 1$ and $\forall i \in [0, \ell-1]$: $m_i \in \{0,1\}$ and $\sum_{i=0}^{\ell-1} m_i = 1$, then add $\mathbf{c}^{(\ell)}$ to $\mathsf{unit\text{-}vec}$.
- Upon receiving $(\textsc{Check}, \mathsf{sid}, \mathbf{c}^{(\ell)})$ from any party $P \in \mathcal{K} \cup \mathcal{U}$:
  - If $\mathbf{c}^{(\ell)}$ is recorded in $\mathsf{unit\text{-}vec}$, then return $(\textsc{Checked}, \mathsf{sid}, \mathbf{c}^{(\ell)}, \mathsf{valid})$ to P.
  - Otherwise, return $(\textsc{Checked}, \mathsf{sid}, \mathbf{c}^{(\ell)}, \mathsf{unknown})$ to P.

Additive Homomorphism:

- Upon receiving $(\textsc{Add}, \mathsf{sid}, (c_1, \ldots, c_\ell))$ from any party $P \in \mathcal{K} \cup \mathcal{U}$:
  - If all $(c_1, m_1), \ldots, (c_\ell, m_\ell)$ are recorded in $\mathsf{ciphertext}$, then do
    * Compute $c := \mathsf{Add}(c_1, \ldots, c_\ell)$ and $m = \sum_{j=1}^{\ell} m_j$
    * Record pair (c, m) in $\mathsf{ciphertext}$; init $J_c := \emptyset$; send $(\textsc{Sum}, \mathsf{sid}, c)$ to P.
- Upon receiving $(\textsc{Scale}, \mathsf{sid}, c, v)$ from any party $P \in \mathcal{K} \cup \mathcal{U}$:
  - If (c, m) is recorded in $\mathsf{ciphertext}$, then do
    * Compute $c' := \mathsf{Scale}(c, v)$ and $m' = m * v$
    * Record pair (c', m') in $\mathsf{ciphertext}$; init $J_{c'} := \emptyset$; and send $(\textsc{Scale}, \mathsf{sid}, c')$ to P.

Decryption:

- Upon receiving $(\textsc{Decrypt}, \mathsf{sid}, c)$ from $\mathsf{K}_i \in \mathcal{K}$, send a notification message $(\textsc{DecNotify}, \mathsf{sid}, \mathsf{K}_i)$ to $\mathcal{A}$.
  - If c has been recorded in $\mathsf{ciphertext}$, set $J_c := J_c \cup \{\mathsf{K}_i\}$.
  - If $|J_c \cap \mathcal{K}_{\mathsf{honest}}| + |\mathcal{K}_{\mathsf{cor}}| \geq t$, and a pair (c, m) has been recorded in $\mathsf{ciphertext}$, then send $(\textsc{DecLeak}, \mathsf{sid}, c, m)$ to $\mathcal{A}$.
  - If $|J_c| = t$ and a pair (c, m) has been recorded in $\mathsf{ciphertext}$, then add (c, m) to $\mathsf{plaintext}$.
  - If c is not recorded in $\mathsf{ciphertext}$, set $J'_c := J'_c \cup \{\mathsf{K}_i\}$ where J'_c is empty initially. If $|J'_c \cap \mathcal{K}_{\mathsf{honest}}| + |\mathcal{K}_{\mathsf{cor}}| \geq t$, then send $(\textsc{DecLeak}, \mathsf{sid}, \langle c, \mathsf{Dec}_{\mathsf{sk}}(c) \rangle)$ to the adversary $\mathcal{A}$.
  - If $|J'_c| = t$, then put $(c, \mathsf{Dec}_{\mathsf{sk}}(c))$ to $\mathsf{plaintext}$.
- Upon receiving $(\textsc{ReadDec}, \mathsf{sid})$ from any party $P \in \mathcal{K} \cup \mathcal{U}$, send $(\textsc{Plaintext}, \mathsf{sid}, \mathsf{plaintext})$ to party P.

Fig. 15: Threshold homomorphic encryption functionality $\mathcal{F}_{\text{THVE}}^{t,k}$

Functionality $\mathcal{G}_{\text{CLOCK}}$

The functionality interacts with a set of parties $\mathbb{P}$, a set of functionalities $\mathbb{F}$, and the adversary $\mathcal{A}$. It is parametrized with variable τ, $\mathbb{P}$, and $\mathbb{F}$.
Initially, set $\tau := 0$, $\mathbb{P} := \emptyset$, and $\mathbb{F} := \emptyset$.

Registration:

- Upon receiving $(\text{REGISTER}, \mathsf{sid})$ from party p, set $\mathbb{P} := \mathbb{P} \cup \{p\}$ and create variable $T_p := 0$.
- Upon receiving $(\text{REGISTER}, \mathsf{sid})$ from functionality $\mathcal{F}$, set $\mathbb{F} := \mathbb{F} \cup \{\mathcal{F}\}$ and create variable $T_{\mathcal{F}} := 0$.
- Upon receiving $(\text{DE-REGISTER}, \mathsf{sid})$ from party p, set $\mathbb{P} := \mathbb{P} \setminus \{p\}$ and remove variable T_p.
- Upon receiving $(\text{DE-REGISTER}, \mathsf{sid})$ from functionality $\mathcal{F}$, set $\mathbb{F} := \mathbb{F} \setminus \{\mathcal{F}\}$ and remove variable $T_{\mathcal{F}}$.
- Upon receiving $(\text{GET-REG}, \mathsf{sid})$ from $\mathcal{A}$, return $(\text{GET-REG}, \mathsf{sid}, \mathbb{P}, \mathbb{F})$ to $\mathcal{A}$.

Synchronization:

- Upon receiving $(\text{TICK}, \mathsf{sid})$ from party $p \in \mathbb{P}$, set $T_p := 1$; Invoke procedure Clock-Update and send $(\text{TICK}, \mathsf{sid}, p)$ to $\mathcal{A}$.
- Upon receiving $(\text{TICK}, \mathsf{sid})$ from functionality $\mathcal{F} \in \mathbb{F}$, set $T_{\mathcal{F}} := 1$; Invoke procedure Clock-Update and send $(\text{TICK}, \mathsf{sid}, \mathcal{F})$ to $\mathcal{F}$.
- Upon receiving $(\text{GETTIME}, \mathsf{sid})$ from any participant, return $(\text{GETTIME}, \mathsf{sid}, \tau)$ to the requester.

Procedure Clock-Update:

- If $T_{\mathcal{F}} = 1$ for all $\mathcal{F} \in \mathbb{F}$ and $T_p = 1$ for all the honest $p \in \mathbb{P}$, then set $\tau := \tau + 1$, and reset $T_{\mathcal{F}} := 0$ for all $\mathcal{F} \in \mathbb{F}$ and $T_p := 0$ for all $p \in \mathbb{P}$.

Fig. 16: Functionality $\mathcal{G}_{\text{CLOCK}}$

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:*
$$\Pr\begin{bmatrix}(x,w) \leftarrow \mathcal{A}; \tau \leftarrow \langle P(x,w), V(x)\rangle : \\ (x,w) \in \mathcal{R}_{\mathcal{L}} \vee \phi(x,\tau) = 1\end{bmatrix} = 1$$
- *(Computational) soundness:*
$$\Pr\begin{bmatrix}x \leftarrow \mathcal{A}; \tau \leftarrow \langle \mathcal{A}, V(x)\rangle : \\ x \notin \mathcal{L} \wedge \phi(x,\tau) = 1\end{bmatrix} = \mathsf{negl}(\lambda)$$

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* Sim *such that*

$$\Pr\begin{bmatrix}(x,w,r) \leftarrow \mathcal{A}; \\ \tau \leftarrow \langle P(x,w), V(x;r)\rangle : \\ (x,w) \in \mathcal{R}_{\mathcal{L}} \wedge \mathcal{A}(\tau) = 1\end{bmatrix} \approx \Pr\begin{bmatrix}(x,w,r) \leftarrow \mathcal{A}; \\ \tau \leftarrow \mathsf{Sim}(x;r) : \\ (x,w) \in \mathcal{R}_{\mathcal{L}} \wedge \mathcal{A}(\tau) = 1\end{bmatrix}$$

Public coin SHVZK proofs/arguments can be transformed to a non-interactive one (in the random oracle model [27]) by using Fiat-Shamir heuristic [28] 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 [29] 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 $f(x_1,\dots,x_n)=0$ evaluated with random $x_1,\dots,x_n \leftarrow \mathbb{Z}_p$ is at most $\frac{d}{p}$.*

Therefore, there are two multi-variate polynomials f_1, f_2. If $f_1(x_1,\dots,x_n)-f_2(x_1,\dots,x_n)=0$ for random $x_1,\dots,x_n \leftarrow \mathbb{Z}_p$, 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 λ.

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 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 valid if and only if $c = g^m h^r$.

Pedersen commitment is also additively homomorphic, i.e.

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

Proof of Theorem 1.

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 that

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

For soundness, first of all, the Sigma protocols for commitments of i_ℓ, $\ell\in[\log n]$ 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 $i_\ell\in\{0,1\}$.

Moreover, $\prod_{j=0}^{n-1}\left((C_j)^{x^{\log n}}\cdot\mathsf{Enc}_{\mathsf{pk}}(-\prod_{\ell=1}^{\log n}z_{\ell,j_\ell};0)\right)^{y^j}$ 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_ℓ is committed, Schwartz-Zippel lemma, $\prod_{j=0}^{n-1}\left((C_j)^{x^{\log n}}\cdot\mathsf{Enc}_{\mathsf{pk}}(-\prod_{\ell=1}^{\log n}z_{\ell,j_\ell};0)\right)^{y^j}\cdot\prod_{\ell=0}^{\log n-1}(D_\ell)^{x^\ell}$ 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 Sim that takes input as the statement $(C_0,\dots,C_{n-1})$ 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, 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

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

After that, Sim outputs the simulated transcript as

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

This concludes our proof.

A.7 Supplementary material for Section 6

Proof of Theorem 2.

Proof. To prove the theorem, we construct a simulator $\mathcal{S}$ such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish between (i) the real execution $\mathsf{EXEC}^{\mathcal{F}_{\text{LEDGER}}, \mathcal{F}_{\text{THVE}}^{t,k}}_{\Pi_{\text{VOTE}}^{t,k,n,m}, \mathcal{A}, \mathcal{Z}}$ where the parties $\mathcal{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$, $\mathcal{E} := \{\mathsf{E}_1, \ldots, \mathsf{E}_m\}$ and $\mathcal{C} := \{\mathsf{C}_1, \ldots, \mathsf{C}_k\}$ run protocol $\Pi_{\text{VOTE}}^{t,k,n,m}$ in the $\{\mathcal{F}_{\text{LEDGER}}, \mathcal{F}_{\text{THVE}}^{t,k}\}$-hybrid world and the corrupted parties are controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\mathsf{EXEC}_{\mathcal{F}_{\text{VOTE}}^{t,k}, \mathcal{S}, \mathcal{Z}}$ where the parties interact with functionality $\mathcal{F}_{\text{VOTE}}^{t,k}$ in the ideal model and corrupted parties are controlled by the simulator $\mathcal{S}$. Let $\mathcal{V}_{\mathsf{cor}} \subseteq \mathcal{V}$, $\mathcal{E}_{\mathsf{cor}} \subseteq \mathcal{E}$ and $\mathcal{C}_{\mathsf{cor}} \subseteq \mathcal{C}$ be the set of corrupted voters, experts and voting committee members, respectively.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ simulates honest voters $\mathsf{V}_i \in \mathcal{V} \setminus \mathcal{V}_{\mathsf{cor}}$, honest experts $\mathsf{E}_i \in \mathcal{E} \setminus \mathcal{E}_{\mathsf{cor}}$, trustees $\mathsf{C}_j \in \mathcal{C} \setminus \mathcal{C}_{\mathsf{cor}}$ and functionalities $\mathcal{F}_{\text{LEDGER}}, \mathcal{F}_{\text{THVE}}^{t,k}$. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

In the preparation phase:

- Upon receiving $(\text{INITNOTIFY}, \mathsf{sid}, \mathsf{C}_j)$ from the external $\mathcal{F}_{\text{VOTE}}^{t,k}$ for an honest voting committee $\mathsf{C}_j \in \mathcal{C} \setminus \mathcal{C}_{\mathsf{cor}}$, the simulator $\mathcal{S}$ acts as C_j, following the protocol $\Pi_{\text{VOTE}}^{t,k,n,m}$ as if C_j receives $(\text{INIT}, \mathsf{sid})$ from the environment $\mathcal{Z}$.

In the ballot casting phase:

- Upon receiving $(\text{VOTENOTIFY}, \mathsf{sid}, \mathsf{E}_i, \beta_i)$ from the external $\mathcal{F}_{\text{VOTE}}^{t,k}$ for an honest expert $\mathsf{E}_i \in \mathcal{E} \setminus \mathcal{E}_{\mathsf{cor}}$, the simulator $\mathcal{S}$, for $\ell \in [0, 2]$: sends $(\text{ENCRYPT}, \mathsf{sid}, 0)$ to $\mathcal{F}_{\text{THVE}}^{t,k}$ and receive $(\text{CIPHERTEXT}, \mathsf{sid}, c_{i,\ell})$ from $\mathcal{F}_{\text{THVE}}^{t,k}$. It then simulates the functionality $\mathcal{F}_{\text{THVE}}^{t,k}$ to add $\mathbf{c_i}^{(3)} := (c_{i,0}, c_{i,1}, c_{i,2})$ to the internal $\mathsf{unit\text{-}vec}$ of $\mathcal{F}_{\text{THVE}}^{t,k}$. The simulator $\mathcal{S}$ then sends $(\text{POST}, \mathsf{sid}, \mathbf{c_i}^{(3)}, \beta_i)$ to $\mathcal{F}_{\text{LEDGER}}$.
- Upon receiving $(\text{CASTNOTIFY}, \mathsf{sid}, \mathsf{V}_j, \alpha_j)$ from the external $\mathcal{F}_{\text{VOTE}}^{t,k}$ for an honest expert $\mathsf{V}_j \in \mathcal{V} \setminus \mathcal{V}_{\mathsf{cor}}$, the simulator $\mathcal{S}$, for $\ell \in [0, m+2]$: sends $(\text{ENCRYPT}, \mathsf{sid}, 0)$ to $\mathcal{F}_{\text{THVE}}^{t,k}$ and receive $(\text{CIPHERTEXT}, \mathsf{sid}, u_{j,\ell})$ from $\mathcal{F}_{\text{THVE}}^{t,k}$. It then simulates the functionality $\mathcal{F}_{\text{THVE}}^{t,k}$ to add $\mathbf{u_j}^{(m+3)} := (u_{j,0}, \ldots, u_{j,m+2})$ to the internal $\mathsf{unit\text{-}vec}$ of $\mathcal{F}_{\text{THVE}}^{t,k}$. The simulator $\mathcal{S}$ then sends $(\text{POST}, \mathsf{sid}, \mathbf{u_j}^{(m+3)}, \alpha_j)$ to $\mathcal{F}_{\text{LEDGER}}$.
- Once the simulated $\mathcal{F}_{\text{LEDGER}}$ receives $(\text{POST}, \mathsf{sid}, \mathbf{c_i}^{(3)})$ from a corrupted expert $\mathsf{E}_i \in \mathcal{E}_{\mathsf{cor}}$, the simulator $\mathcal{S}$ checks the internal state of $\mathcal{F}_{\text{THVE}}^{t,k}$ for recorded $\{(c_{i,\ell}, e_{i,\ell})\}_{\ell \in [0,2]}$ from $\mathsf{ciphertext}$. It then computes $v_i \leftarrow \mathsf{decode}(e_{i,0}, \ldots, e_{i,2})$ and sends $(\text{VOTE}, \mathsf{sid}, v_i)$ to $\mathcal{F}_{\text{VOTE}}^{t,k}$.
- Once the simulated $\mathcal{F}_{\text{LEDGER}}$ receives $(\text{POST}, \mathsf{sid}, \mathbf{u_j}^{(m+3)}, \alpha_j)$ from a corrupted voter $\mathsf{V}_j \in \mathcal{V}_{\mathsf{cor}}$, the simulator $\mathcal{S}$ checks the internal state of $\mathcal{F}_{\text{THVE}}^{t,k}$ for recorded $\{(u_{j,\ell}, e_{j,\ell})\}_{\ell \in [0,m+2]}$ from $\mathsf{ciphertext}$. It then computes $v_j \leftarrow \mathsf{decode}(e_{j,0}, \ldots, e_{j,m+2})$ and sends $(\text{CAST}, \mathsf{sid}, v_j, \alpha_j)$ to $\mathcal{F}_{\text{VOTE}}^{t,k}$.

In the tally phase:

- Upon receiving $(\text{DELCALNOTIFY}, \mathsf{sid}, \mathsf{C}_j)$ from the external $\mathcal{F}_{\text{VOTE}}^{t,k}$ for an honest trustee $\mathsf{C}_j \in \mathcal{C} \setminus \mathcal{C}_{\mathsf{cor}}$, the simulator $\mathcal{S}$ acts as C_j, following the protocol $\Pi_{\text{VOTE}}^{t,k,n,m}$ as if C_j receives $(\text{DELCAL}, \mathsf{sid})$ from $\mathcal{Z}$.

– Upon receiving $(\textsc{TallyNotify}, \mathsf{sid}, \mathsf{C}_j)$ from the external $\mathcal{F}^{t,k}_{\mathrm{Vote}}$ for an honest trustee $\mathsf{C}_j \in \mathcal{C} \setminus \mathcal{C}_{\mathsf{cor}}$, the simulator $\mathcal{S}$ acts as C_j, following the protocol $\Pi^{t,k,n,m}_{\mathrm{Vote}}$ as if C_j receives $(\textsc{Tally}, \mathsf{sid})$ from $\mathcal{Z}$.
– Upon receiving $(\textsc{Leak}, \mathsf{sid}, \tau)$ from the external $\mathcal{F}^{t,k}_{\mathrm{Vote}}$, the simulator $\mathcal{S}$ parses τ as (τ_0, τ_1, τ_2) and acts as the simulated $\mathcal{F}^{t,k}_{\mathrm{THVE}}$ to send $(\textsc{DecLeak}, \mathsf{sid}, x_\ell, \tau_\ell)$, $\ell \in [0, 2]$, where x_ℓ are the tally ciphertexts received by $\mathcal{F}^{t,k}_{\mathrm{THVE}}$ for final decryption.

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \ldots, \mathcal{H}_2$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathsf{EXEC}^{\mathcal{F}_{\mathrm{Ledger}}, \mathcal{F}^{t,k}_{\mathrm{THVE}}}_{\Pi^{t,k,n,m}_{\mathrm{Vote}}, \mathcal{A}, \mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except the simulator $\mathcal{S}$ internally simulates $\mathcal{F}^{t,k}_{\mathrm{THVE}}$. For all the honest voters and experts, $\mathcal{S}$ acts as the party to execute the protocol according to the description as if the voter receives command $(\textsc{Cast}, \mathsf{sid}, \perp, \alpha_j)$ or the expert receives $(\textsc{Vote}, \mathsf{sid}, \perp)$ from the environment $\mathcal{Z}$, where $\perp$ stands for blank ballot, which is represented by a zero vector.

Claim: $\mathcal{H}_1$ and $\mathcal{H}_0$ are indistinguishable.

Proof. The probability that any adversary $\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}$ can distinguish $\mathcal{H}_1$ from $\mathcal{H}_0$ is bounded by $\mathsf{AdvCPA}_{\mathcal{A}}(1^\lambda)$. More specifically, we now show the if there exists an adversary $\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}$ who can distinguish $\mathcal{H}_1$ from $\mathcal{H}_0$, then we can construction an adversary $\mathcal{B}$ that can break the IND-CPA game of the underlying threshold Public key encryption by reduction. During the IND-CPA game, $\mathcal{B}$ receives a public key pk^* from the challenger. There must be at least one honest trustee in this case, and with our loss of generality, assume C_x is honest. During the preparation phase, $\mathcal{B}$ posts pk^* as C_x's public key together with simulated proof. During the ballot casting phase, for each honest voter V_i, $i \in [n]$, $\mathcal{B}$ sends $m_0 := 0$ and $m_1 := v_i$ to the IND-CPA challenger, and receives c^*. It posts c^* as the honest voter's encrypted ballot. It is easy to see that, when c^* encrypts m_0, the adversary's view is indistinguishable from $\mathcal{H}_1$; when c^* encrypts m_1, the adversary's view is indistinguishable from $\mathcal{H}_0$. Hence, if $\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}$ can distinguish $\mathcal{H}_1$ from $\mathcal{H}_0$ with non-negligible probability, then $\mathcal{B}$ can break the IND-CPA game with the same probability.

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except the following: During the tally phase, the simulator $\mathcal{S}$ acts as $\mathcal{F}^{t,k}_{\mathrm{THVE}}$ to answer all the decryption queries such that it is consistent with the output of $\mathcal{F}^{t,k}_{\mathrm{Vote}}$.

Claim: $\mathcal{H}_2$ and $\mathcal{H}_1$ are indistinguishable.

The adversary's view of $\mathcal{H}_2$ is identical to the simulated view $\mathsf{EXEC}_{\mathcal{F}^{t,k}_{\mathrm{Vote}}, \mathcal{S}, \mathcal{Z}}$. Therefore, no PPT $\mathcal{Z}$ can distinguish the view of the ideal execution from the view of the real execution with more than negligible probability.

A.8 Consensus background

The overarching aim of all decision-making mechanisms is to reach the best decision. However, it is usually unclear what constitutes the best alternative. In other words, in a multi-party decision-making process, it is difficult to agree on what constitutes the best solution, due to differences in individual preferences,interests, knowledge, skill, orientation, etc. Therefore, integration of community-wide knowledge, skills and expertise of members is fundamental for long-term sustainability (especially for blockchain developmental projects).

Consensus building [45] has been identified as a way to deal with complex, strategic and often controversial planning and decision-making. Sustained innovation and development requires the continued maintenance of the complex interaction between all stakeholders (with varying expertise, skill sets and values). The goal is to adopt and implement solutions that offers “mutual” gains among contending stakeholders. Consensus building or processes are typically time-costly and resource-consuming. However, blockchain infrastructure and nature of blockchains mitigates against these potential drawbacks. For instance, by design, treasury system planning supports lengthy discussions and deliberations over the range of treasury epochs available in any single treasury period. Furthermore, the decentralised nature of blockchain technologies reduces the monetary costs associated with collaborative decision making.

Critics of collaborative decision-making argue that, perhaps, consensus kills innovation, creativity and uniqueness. They suggest that individuals that participate in the process tend to abandon their decisions, so as to align with the rest of the group. However, this is not necessarily true considering the negotiation (debates) and interaction that takes place before compromises are made or decisions are reached. Consensus empowers the individual through social activity and interaction, rather than suppress the individual [46].

One of the goals of collaborative decision-making (community-inclusive participation) is improved community relations, and research evidence shows that engaged citizenry/community is better than a passive one [47]. As a result, citizens become willing evaluators of decisions and policies, which results in improved community-wide support for decisions reached. Thus, making governance, which is particularly important to blockchain systems, easy.

Agreement, perhaps, is the single most popular criterion for evaluating consensus. Typically in the literature, consensus is analysed in terms of agreement, and is not *Majority Rule*. Consensus involves the evaluation of the agreement among a set of parties on a set of alternative solutions [48] in a multi-party collaborative decision-making process (e.g., the treasury system decision-making process). Classical definitions of consensus imply absolute agreement among all parties as a condition for consensus. This type of (full) consensus is quite feasible within small teams or organisations with members having relevant information needed for decision-making. However, the extremely low possibility of achieving this, makes this definition problematic and less useful. By extension, this definition would equate the utility of decisions with "almost unanimous" (i.e., very high but not perfect) agreement to those with complete disagreement - as both not being useful.

Nonetheless, in real-world scenarios multi-party decisions need not be unanimous or in absolute agreement (full consensus) for decisions to be useful. Therefore, in order to accommodate the spectrum of consensus between full/unanimous agreement and total disagreement, [48] defined soft agreement as an iterative dynamic process that evaluates the agreement between all participants, and the agreement between the individual participant's preference and the group solution. Typically, two related measures, *Consensus measure* (measure of agreement among all participants) and *Proximity measure* (measure of agreement between individual solutions and collective solution) [48,45] are used to evaluate consensus.

Regardless of the quality of agreement achieved/reached, the outcome of a flawed *process* lacks credibility, is less likely to receive widespread support and would likely result in more tensions among member stakeholders. Well designed consensus processes that involves every stakeholder, regardless of the amount of stake they hold, is likely to produce fair outcomes [45] and receive community-wide acceptance.

In line with the aim of collaborative decision-making, we note that our key goal of evaluating consensus is not to produce "winners and losers", rather, the goal is to forge, enhance and encourage community-wide participation and acceptance (sense of responsibility and belonging) and ownership of the growth, changes and developments of the underlying blockchain system. Therefore, feedback is a key component of consensus evaluation. Information obtained from proximity measure is useful for influencing discussion and minimising disagreement among stakeholders [48].

Evidently, developments or changes with greater consensus are more durable and sustainable because high consensus implies a higher agreement (support) among the stakeholders of the decision making process. Furthermore, agreements of this nature tend to be of very high quality because they take into consideration the knowledge (rather than interest alone) offered by each stakeholder [45].

With the help of an illustrative example of a treasury period, we now present an evaluation of consensus of the treasury system decision-making process.

A.9 Example treasury consensus evaluation

Typically, consensus is measured through the use of some dissimilarity function e.g., cosine of angles, Euclidean distance, etc. between corresponding individual preferences/solutions (proximity measure) and group solution, as well as the evaluation of agreement among all participants on the final/group solution (consensus measure).

For the purpose of consensus measurement in our treasury system decision-making, we propose an adaptation of the approach in [49] which is itself is an adaptation of [48] in order to accommodate nuances peculiar to blockchain systems (or cryptocurrencies), e.g., cryptocurrency stake distribution. Specifically, for treasury system consensus measurement, we identify the following key elements:

– Proposals (or alternative solutions)
– Stake holders in the system (or participants in the decision-making process)
– Treasury funds (or available resources)

– The decision-making process (or voting scheme)
– Outcomes of the decision-making process (or solution set)
– Agreement among all participants (or consensus measure)
– Agreement between individual solutions and the treasury outcome/solution (or proximity measure)

Additionally, we highlight the processes for consensus evaluation in the treasury system collaborative decision-making scheme:

– Preference specification
– Collective solution calculation
– Distance measure
– Distance aggregation
– Consensus measure and
– Proximity measure

We now present each of the various stages involved in the consensus evaluation process.

List of titles of proposals requesting funds

– **Proposal 1 :** Purchase and Installation of ATMs for the cryptocurrency
– **Proposal 2 :** Facilitation of listing of cryptocurrency on major cryptocurrency exchanges
– **Proposal 3 :** Cryptocurrency awareness and advertisement campaigns boost cryptocurrency exchange rate
– **Proposal 4 :** Creation of online educational resources such as YouTube videos, and general training for cryptocurrency trading
– **Proposal 5 :** Construction of cryptocurrency international headquarters and liaison office on every continent
– **Proposal 6 :** Development of third-party software and tools to support cryptocurrency, e.g., establishment of a new peer-to-peer mining pool software
– **Proposal 7 :** Establishment of a cryptocurrency legal team
– **Proposal 8 :** Organisation of a cryptocurrency conference
– **Proposal 9 :** Analysis of cryptocurrency protocol security and proofs
– **Proposal 10 :** Organisation of an annual dinner party and award night for community members

Participation information

We assume 5 people (2 experts and 3 voters) are involved in the current treasury voting process. For simplicity, without the loss of generality, we also assume a flat model of stake distribution in this illustrative treasury period. That is, we assume that all participants (expert/voter) have equal stake in the system (e.g., 1 cryptocoin).

Preference specification

Each participant respectively specify his preferences based on his assessment of the individual proposals, using criteria/guidelines such as: usefulness of proposal, timeliness, cost-benefit impact of proposal, profile of proposer, relevance of project, urgency of proposal, amount of funds requested, duration of project, team in charge of project, quality of proposal, etc. However, users are free to further evaluate proposals as they deem fit (or based on their personal judgment).

Particularly, users vote YES, NO, ABSTAIN for proposals either directly or indirectly by delegating their voting power to experts in that particular area. We encode a YES, NO, or ABSTAIN votes as $1, 0$, or $\perp$ respectively. We remark that the ballots of users who vote $\perp$ for any proposal are treated as being the same as the treasury outcome for that proposal. Hence, for consensus evaluation, they are considered as being in agreement with whatever the outcome of the affected proposal is.

Collective solution calculation

For voting on the treasury, the group solution is obtained through the application of the voting rule (e.g., majority voting, fuzzy threshold voting) on the ballot casted by the experts and voters. Specifically, for our treasury system, where the voting rule is *Fuzzy Threshold Voting*, this corresponds to ranking of the alternative proposals based on the number of votes for minus the number of votes against, and checking that the remainder is at least 10% of all votes recorded. Thereafter, winning votes are determined as those that receive funding from the (ranked) list of all "qualified" proposals. Therefore, proposals that meet the minimum threshold but do not receive funding because of the limitation of available funds (and their relatively low overall position in the ranked list) are not considered members of the set of "winning proposals". The set of "winning projects " are those who will receive funding according to the decision reached on the treasury system (voting result). Table 1 provides information on how participants casted their ballots.

Table 1: User preference

	Project 1	Project 2	Project 3	Project 4	Project 5	Project 6	Project 7	Project 8	Project 9	Project 10
User 1	1	B	1	B	1	B	A	A	B	⊥
User 2	⊥	1	1	⊥	0	1	1	0	⊥	1
User 3 / Expert A	1	1	1	0	0	1	0	0	1	0
User 4 / Expert B	1	1	0	1	0	1	0	1	1	0
User 5	0	1	A	1	⊥	A	1	⊥	A	1
Total	3	5	4	3	1	5	2	1	4	2
Treasury Decision	0	1	1	0	0	1	0	0	1	0

(Normalised) distance measure

For each participant, we calculate distance measure (DM) of every vote/ballot for each project by comparing the participant's choice with the treasury system funding decision. We apply the dissimilarity function below:

$$DM_{i,j} = |PP_{i,j} - TS_j|$$

$$NDM_{i,j} = \frac{DM_{i,j}}{|PS|}$$

where $DM_{i,j}$ (respectively, $NDM_{i,j}$) is the distance (respectively, normalised distance) between the i^{th} participant's choice for project j and the treasury solution for project j is TS_j. $PP_{i,j}$ is the i^{th} *participant's preference/choice* for project j and PS is the *preference size* of voter's choice. In the case of our treasury system $PS = 2$, for YES and NO, because ABSTAIN is handled differently. As earlier explained, for consensus evaluation, the choice of voters/experts who vote ABSTAIN i.e. ⊥ are considered as being the same as treasury funding decision for any particular project. Hence, a distance measure of zero(0) is assigned for a participant who votes ABSTAIN for any project proposal. The distance measure for the treasury system decision making is presented in Table 2.

Table 2: Distance specification

	Project 1	Project 2	Project 3	Project 4	Project 5	Project 6	Project 7	Project 8	Project 9	Project 10
User 1	1	0	0	1	1	0	0	0	0	0
User 2	0	0	0	0	0	0	1	0	0	1
User 3 / Expert A	1	0	0	0	0	0	0	0	0	0
User 4 / Expert B	1	0	1	1	0	0	0	1	0	0
User 5	0	0	0	1	0	0	1	0	0	1

Distance aggregation and consensus degree on projects

Using the normalised distance measure, NDM, we calculate the degree of consensus among all participants (voters and experts) on each project as follows:

$$CD_j = 1 - \sum_{i=1}^{p} \frac{NDM_{i,j}}{p}$$

Table 3: Consensus degree

	Project 1	Project 2	Project 3	Project 4	Project 5	Project 6	Project 7	Project 8	Project 9	Project 10
User 1	0.5	0	0	0.5	0.5	0	0	0	0	0
User 2	0	0	0	0	0	0	0.5	0	0	0.5
User 3 / Expert A	0.5	0	0	0	0	0	0	0	0	0
User 4 / Expert B	0.5	0	0.5	0.5	0	0	0	0.5	0	0
User 5	0	0	0	0.5	0	0	0.5	0	0	0.5
Total	1.5	0	0.5	1.5	0.5	0	1.0	0.5	0	1.0
Consensus degree	0.7	1.0	0.9	0.7	0.9	1.0	0.8	0.9	1.0	0.8

where CD_j is the consensus degree for project j, and p is the size/number of participants.

Table 4: Consensus measure for various values of β

β	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CM	0.87	0.881	0.891	0.902	0.912	0.923	0.933	0.944	0.954	0.964	0.975

Table 5: Proximity measure for $\beta = 0.8$

	Proj. 1	Proj. 2	Proj. 3	Proj. 4	Proj. 5	Proj. 6	Proj. 7	Proj. 8	Proj. 9	Proj. 10	Avg.	PM_i	$PM_i(\beta 0.8)$
User 1	0.5	1	1	0.5	0.5	1	1	1	1	1	0.85	$(1-\beta)0.85+\beta$	0.97
User 2	1	1	1	1	1	1	0.5	1	1	0.5	0.9	$(1-\beta)0.9+\beta$	0.98
User 3/Exp. A	0.5	1	1	1	1	1	1	1	1	1	0.95	$(1-\beta)0.95+\beta$	0.99
User 4/Exp. B	0.5	1	0.5	0.5	1	1	1	0.5	1	1	0.8	$(1-\beta)0.8+0.875\beta$	0.86
User 5	1	1	1	0.5	1	1	0.5	1	1	0.5	0.85	$(1-\beta)0.85+\beta$	0.97

Consensus measure

We now proceed to calculate overall consensus among all participants (for our example treasury period decision under review) through an aggregation of the consensus degrees, CD. The consensus measure, CM, is calculated through the aggregation procedure of [50], which utilises Ordered Weighted Average (OWA). The goal of the aggregation operator is to enable the consensus degrees on the "winning projects" have more importance or weight [48] in the aggregation procedure. The expression for CM is as follows:

$$CM = (1-\beta) \cdot \sum_{i=1}^{n} \frac{CD_i}{n} + \beta \cdot \sum_{s=1}^{t} \frac{CDW_s}{t}$$

where CDW is the set of consensus degrees for "winning projects", t is its cardinality, n is the total number of projects, and $\beta \in [0,1]$ is used to control the influence of consensus degree of the winning projects on the overall consensus measure in the treasury system. Clearly, higher values of β causes consensus degree of the winning projects to highly influence the overall consensus measure. Typical values of β recommended in the literature are $0.7, 0.8$, and 0.9 [48,49]. We use a β value of 0.8 in our treasury system to emphasize the importance of consensus degree among the participants on the winning projects. Table 4 shows the consensus measure for different values of β.

Proximity measure of each participant

Here, we evaluate the proximity measure of each participant's voting preference to the collective treasury (funding) decision by aggregating each participant's distance measure across all the projects. Similar to the calculation of the consensus measure, we utilise OWA aggregation operator as follows:

$$PM_i = (1-\beta)\frac{\sum_{j=1}^{p}(1-NDM_{i,j})}{p} + \beta\Big(1 - \frac{\sum_{k=1}^{t} NDMW_{i,k}}{t}\Big)$$

where $NDM_{i,j}$ is the distance measure between participant $i's$ preference for project j and the treasury (collective) decision for project j. For participant i, $NDMW_{i,k}$ is a special normalised distance measure between the collective decision for project k in the "set of winning projects " and the corresponding participant $i's$ preference for that project. That is, NDMW only considers normalised distance measures for projects that receive funding a.k.a winning projects.

As earlier explained, we assign a value of 0.8 to β and present the proximity measure between each participant's preference and the treasury funding decision in Table 5.

Evidently, participants with high proximity measures contribute positively towards the treasury system consensus while proximity measures close to zero signify negative contribution towards overall treasury system consensus. Additionally, it can be observed that User 1 and User 5 both have the same proximity measure of 0.97, despite having different voting preferences. However, this is so because the two users voted exactly the same way for projects in the "winning set", and the "winning set" or collective decision highly influenced our aggregated measures due to the high value of β used.