# A Treasury System for Cryptocurrencies: Enabling Better Collaborative Intelligence*

Bingsheng Zhang[1], Roman Oliynykov[2], Hamed Balogun[3], and Tamara Finogina[4]

[1] Lancaster University, UK
b.zhang2@lancaster.ac.uk
[2] IOHK
and V.N.Karazin Kharkov National University, Ukraine
roman.oliynykov@iohk.io
[3] Lancaster University, UK
h.balogun@lancaster.ac.uk
[4] Scytl R & D, Spain
tamara.finogina@skolkovotech.ru

**Abstract.** A treasury system is a community controlled and decentralized collaborative decision-making mechanism for sustainable funding of the underlying 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 security 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 voters can either vote directly on the proposed projects or delegate their votes to experts. Its core component is a universally composable secure end-to-end verifiable online 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 under Java/Scala, 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

Since the rise of Bitcoin [1] in 2008, hundreds of cryptocurrencies have been developed and in circulation. At the time of writing, the total market capitalization of the Top 15 cryptocurrencies exceeds $150 billion [2]. An increasing number of companies, as well as traditional financial institutions and governments have expressed their interests in adopting blockchain technologies [3]. As a security critical software, the typical blockchain development lifecycle requires high level of stable and reliable funding. On the other hand, the decentralized architecture of blockchain technologies makes it hard to have a centralized control of the funding for secure development processes. Absence of a centralized control over the operation process is a key feature expected from the blockchain systems. It can be ensured with decentralization and honest majority governance. Among other prefered features, it provides better securiy guarantees due to the elimination of single point of failure; in most cases, such systems may also implement extra transformations for end user privacy. Moreover, most existing blockchain development organizations rely on ad-hoc donation, which is not sustainable in the long term.

In short-term perspective, a cryptocurrency operation requires to have a decentralized honest majority of nodes that supports a distributed ledger maintenance. In the middle- and long-term perspective, an additional factor is crucial. Any cryptocurrency exists as a complex technical system that requires continuous designing and implementation of multiple complex protocols and algorithms that, in turn, requires the appropriate funding level. Lack of funding would lead to cryptocurrency operation freeze, and to a deficient issue resolution routine. Occasional funding with weakly managed voluntary contributions may result in security vulnerabilities and system instability that would affect the exchange rate and lead to dramatic halt of its market extension. Thus, stable funding for the system support and development is a key to the cryptocurrency middle and long term market outlook.

We may consider different options to provide the needed funding. It can be money raised from venture funds which are interested in a potential of the blockchain technology, including ICOs, or it can be community donations raised through the crowdfunding process. Another approach is a cryptocurrency self-sufficiency to ensure sustainable funding implemented, e.g., the Dash governance system [4,5]. Our Treasury approach allows to get stable and reliable funding for cryptocurrency development and support of its security level under transparent and verifiable control of stakeholder community. Constructing the whole procedure on the blockchain gives great benefits over simple message exchange and keeping a separate voting database. It provides much higher level of the general transparency and verifiability of the system, as well as allows full verification without need of trusted parties.

A treasury system can simply be described as a “self-sustenance” mechanism. Concepts encompassed by this self-sustenance include development, maintenance and advancement of the underlying system or cryptocurrency. In blockchain technologies, a treasury system serves as a decentralised means of achieving and ensuring continued existence and improvement of its parentsystem (e.g., cryptocurrency) by providing a regular source of funding for tasks necessary for the survival and success of the system. These tasks include marketing, software development, legal fees, advertising and publicity, etc. The treasury system is to provide a means of proposal submission, links/URLs to detailed information about proposals, voting mechanism for deciding on proposals, a sustainable means of funding/treasury (regardless of whether the underlying cryptocurrency has money supply cap or not).

### 1.1 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. Every treasury period, stake holders in the system submit proposals for projects to be funded. Due to shortage in the amount of funds available for funding, only a number of these proposed projects can be funded. Thus, a collective decision-making system is required to decide on the projects that will be funded. To this end, our treasury system contains a voting scheme using distributed key generation and threshold cryptography. In order to guarantee verifiability, integrity, privacy and overall security of the system, several design choices were made. Justifications and discussions on these choices are provided in later sections of this work.

An election rule of voting system provides the method of gathering voters' opinions and selecting the winner or a set of them. There is a big variety of voting systems: plurality voting, preferential voting, cardinal voting, etc. [6]. They have different approaches and properties, but all of them try to archive the goal of satisfying as much as possible voters with election results. Strategical behavior of voters may have to stepping down their direct preferences (e.g., supporting less prefereble proposal due it has higher chances comparing to the most prefereble one, not having both rejected). Proper selection of the election rule allows maximizing amount of voters satisfied of voting results as well as minimize voter's effort on expressing his intent in her ballot (not requiring pairwise comparison of all candidates, etc.).

In a **plurality voting system** each participant is allowed to vote for only one proposal. The most popular one (which polls more votes than any other, or a plurality) is elected [6]. Plurality rule is also known as a simple majority (not absolute majority) rule. However, if there are more than two alternatives, the plurality voting has problems [7], including losing information on true voter preferences (the best strategy is to vote for one of two most popular proposals, not for her favorite one), dispersion of votes across similar proposals, etc. For treasury that requires several winners, this widely spread political election rule is less suitable comparing to others.

**Preferential or ranked voting** describes certain voting systems in which voters rank outcomes in a hierarchy on the ordinal scale. When choosing among more than two options, preferential voting systems provide a number of advantages over plurality voting. One of the main advantages is that election results will more accurately reflect the level of support for all candidates.

There are many preferential voting systems: Instant-runoff voting (IRV) [8], Borda count [9], Single transferable vote [10], Schulze method [11], an optimal single-winner preferential voting system (the GT system) based on optimal mixed strategies computation [12] etc.

The analysis in [13] revealed several defects of preferential voting (e.g., adding an outsider to the voting list may change results on election favorites, etc.) Besides it, practical application of this election rule in treasury may be too complex for voters due too much effort on ranking of tens or even hundreds of proposals.

**Approval voting** is a voting method that allows voters to approve any number of proposals [14]. Winner(s) are chosen by the largest number of supporting ballots. Approval voting are well suitable to multi-winner (at-large) elections.

Approval voting has a number of advantages [15]: simple, quick and easy-to-understand voting process, better expressing true voter intent in her ballot, etc.

Further natural development of approval voting leds to "Yes-No-Abstain" option for each proposal. This scheme is used Dash Governance System [16], The DAO [17], The Fermat Project [18] and other solutions for cryptocurrencies. Recent theoretical analysis of this election rule with variable number of winners, called **Fuzzy threshold voting**, [19] shows advantages of this voting scheme for treasury application.

Thus, in proposed treasury system each particiant can vote "Yes-No-Abstain" arbitrary number of proposals (or delegate this right to a corresponding expert). Besides it, a number of winners is dynamic and limited by treasury funding reward.

## 1.2 Related work

Liquid democracy (also known as delegative democracy [20]) 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 [21]. Although the advantage of liquid democracy has been widely discussed in the literature [22,23,24,25,26], 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 [27] 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 [28], LiquidFeedback [29], Adhocracy [30], GetOpinionated,[31] also offer poor privacy guarantees. It is worth mentioning that Sovereign [32] is a blockchain-based voting protocol for liquid democracy; therefore, its privacy is inherited from the underlying blockchain, which provides pseudonymity-based privacy. Wasa2il [33] is able to achieve End-to-End verifiability because this foils privacy. The best known liquid democracy and proxy democracy voting schemes are nVotes [34] and Statement Voting [35]. 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.

Beyond voting, the Dash Governance System (DGS) [4,5], is the first self-sustenace/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.

Our work differs from these earlier work 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 associated with a treasury system for blockchains. In a worse case scenario, our privacy guarantees are equivalent to that obtainable in the DGS.

### 1.3 Our contribution

In this work, we throughly study the security and design of a treasury system. Here, we highlight the major contributions of this work. We provide the first provably secure blockchain-based treasury system (in the UC-Model) that supports liquid democracy/delegative voting. In a worst case scenario (where a majority or all of the election committee members are malicious), the protocol still provides similar privacy guarantees as that of (best case) Dash Governance System.

A further contribution (of independent interest for interested readers) of this work is the proof of unit vector encryption. The work includes a zero-knowledge proof of unit vector encryption with logarithmic size communication. We also provide details of benchmark implementations and results of tests on the real-world implementation.

## 2 System model

### 2.1 Entities Involved in the Treasury System

- The voting committees $\mathcal{C} := \{\mathsf{C}_1, \ldots, \mathsf{C}_\ell\}$ are a set of nodes that are responsible of generating the voting public key and opening the voting result. They are randomly elected at the beginning of each treasury epoch. (cf. Section 5, below.)
- The voters $\mathcal{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$ are a set of nodes that deposit at least $\alpha$ coins in priori to the voting epoch to participate the voting. The voting power of voter $\mathsf{V}_i$ is proportional to its deposited amount, denoted as $\alpha_i$.
- The experts $\mathcal{E} := \{\mathsf{E}_1, \ldots, \mathsf{E}_m\}$ are a subset of voters that want to receive delegation. Each expert must deposit at least $\beta$ coins in priori to the voting epoch.

## 2.2 Security Model

In this work, we model the security of the treasury system 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* (*ITI*s), 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 probabilistic polynomial time (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.

**Ideal world Execution.** In the ideal world, the voting committees $\{\mathsf{C}_1, \ldots, \mathsf{C}_\ell\}$, the voters $\{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$, and the experts $\{\mathsf{E}_1, \ldots, \mathsf{E}_m\}$ only communicate with an ideal functionality $\mathcal{F}_{\text{VOTE}}$ during the entire election. The ideal functionality $\mathcal{F}_{\text{VOTE}}$ accepts a number of commands from $\{\mathsf{C}_1, \ldots, \mathsf{C}_\ell\}$, $\{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$ and $\{\mathsf{E}_1, \ldots, \mathsf{E}_m\}$. 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. 1, 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 $(\text{INIT}, sid)$ to the ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k}$. The voting will not start until all the voting committees have participated the preparation phase.

*Voting/Delegation phase.* During the voting/delegation phase, the expert $\mathsf{E}_i \in \mathcal{E}$, who has deposited $\beta_i$ coins, can vote for his choice $v_i$ by sending $(\text{VOTE}, sid, v_i, \beta_i)$ to the ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,k}$. Note that $\beta_i$ is a public input and it is always leaked to the adversary $\mathcal{S}$; whereas, 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 has deposited $\alpha_j$ coins, can either vote directly for his choice $v_j$ or delegate his voting power to an expert $\mathsf{E}_i \in \mathcal{E}$. Similarly, when all the voting committees are corrupted, $\mathcal{F}_{\text{VOTE}}^{t,k}$ 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. 3.

## 3 Preliminaries

### 3.1 Notations

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

### 3.2 Zero-knowledge proofs/arguements

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

**Definition 1.** *We say $(P, V)$ is a perfectly complete proof/argument for an NP relation $\mathcal{R}_{\mathcal{L}}$ if for all non-uniform PPT interactive adversaries $\mathcal{A}$ it satisfies*

– *Perfect completeness:*

$$\Pr\left[\begin{array}{l}(x, w) \leftarrow \mathcal{A}; \tau \leftarrow \langle P(x, w), V(x) \rangle : \\ (x, w) \in \mathcal{R}_{\mathcal{L}} \vee \phi(x, \tau) = 1\end{array}\right] = 1$$

– *(Computational) soundness:*

$$\Pr\left[\begin{array}{l}x \leftarrow \mathcal{A}; \tau \leftarrow \langle \mathcal{A}, V(x) \rangle : \\ x \notin \mathcal{L} \wedge \phi(x, \tau) = 1\end{array}\right] = \mathsf{negl}(\lambda)$$

**Functionality $\mathcal{F}^{t,k}_{\textbf{Vote}}$**

The functionality $\mathcal{F}^{t,k}_{\textsc{Vote}}$ interacts with a set of voting committees $\mathcal{C} := \{\mathsf{C}_1, \dots, \mathsf{C}_k\}$, a set of voters $\mathcal{V} := \{\mathsf{V}_1, \dots, \mathsf{V}_n\}$, a set of experts $\mathcal{E} := \{\mathsf{E}_1, \dots, \mathsf{E}_m\}$, and the adversary $\mathcal{S}$. It is parameterized by a delegation calculation algorithm $\mathsf{DelCal}$ (described in Fig. 2) and a tally algorithm $\mathsf{TallyAlg}$ (described in Fig. 3) and variables $\phi_1, \phi_2$, $\tau$, $J_1, J_2, J_3$, $T_1$ and $T_2$. Denote $\mathcal{C}_{\mathsf{cor}}$ and $\mathcal{C}_{\mathsf{honest}}$ as the set of corrupted and honest voting committees, respectively.

Initially, $\phi_1 = \emptyset$, $\phi_2 = \emptyset$, $\tau = \emptyset$, $J_1 = \emptyset$, $J_2 = \emptyset$, and $J_3 = \emptyset$.

**Preparation:**

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

**Voting/Delegation:**

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

**Tally:**

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

Fig. 1: The ideal functionality $\mathcal{F}^{t,k}_{\textsc{Vote}}$

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

**The delegation calculation algorithm** $\mathsf{DelCal}$

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

**Init:**

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

**Delegation interpretation:**

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

**Output:**

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

Fig. 2: The delegation calculation algorithm

**The tally algorithm** $\mathsf{TallyAlg}$

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

**Init:**

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

**Tally Computation:**

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

**Output:**

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

Fig. 3: The tally algorithm

**Definition 2.** *We say a public coin proof/argument $(P, V)$ is a perfect special honest verifier zero-knowledge (SHVZK) for a NP relation $\mathcal{R}_{\mathcal{L}}$ if there exists a PPT simulator* $\mathsf{Sim}$ *such that*

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

Public coin SHVZK proofs/arguments can be transformed to a non-interactive one (in the random oracle model [36]) by using Fiat-Shamir heuristic [37] where a cryptographic hash function is used to compute the challenge instead of having an online verifier.

### 3.3 Schwartz-Zippel lemma

For completeness, we recap a variation of the Schwartz-Zippel lemma [38] that will be used in proving the soundness of the zero-knowledge protocols.

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

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

## 4 Building blocks

### 4.1 Additively homomorphic encryption

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

- $\mathsf{Gen}^{\mathsf{E}}(\mathsf{param})$: pick $\mathsf{sk} \leftarrow \mathbb{Z}_q^*$ and set $\mathsf{pk} := h = g^{sk}$, and output $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}_{\mathsf{pk}}(m; r)$: output $e := (e_1, e_2) = (g^r, g^m h^r)$.
- $\mathsf{Add}(c_1, \ldots, c_\ell)$: output $c := (\prod_{i=1}^{\ell} c_{i,1}, \prod_{i=1}^{\ell} c_{i,2})$.
- $\mathsf{Scale}(c, v)$: output $c^* := ((c_1)^v, (c_2)^v)$.
- $\mathsf{Dec}_{\mathsf{sk}}(e)$: output $\mathsf{Dlog}(e_2 \cdot e_1^{-\mathsf{sk}})$, where $\mathsf{Dlog}(x)$ is the discrete logarithm of $x$. (Note that since $\mathsf{Dlog}(\cdot)$ is not efficient, the message space should be a small set.)

### 4.2 Threshold homomorphic vector encryption

In our treasury system, we use the threshold version of the aforementioned additively homomorphic encryption. The primitive is modelled as an ideal functionality $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ as depicted in Figure 4. It is parameterized by the additively homomorphic encryption algorithms $\mathsf{HE} = (\mathsf{Gen}^{\mathsf{E}}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Scale}, \mathsf{Dec})$. 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 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}^{\mathsf{E}}(\mathsf{param})$. 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 $\mathsf{valid}$; otherwise, it returns $\mathsf{unknown}$. Moreover, the user can run homomorphic operations $\mathsf{Add}$ and $\mathsf{Scale}$ on the ciphertexts if they were encrypted under $\mathsf{pk}$. 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.

### 4.3 Pedersen commitment

In the unit vector zero-knowledge proof, we use Pedersen commitment as a building block. It shares the same group parameters, $\mathsf{param}$, as the lifted ElGamal encryption. It is perfectly hiding and computationally binding under the discrete logarithm assumption. More specifically, it consists of the following 4 PPT algorithms.

- $\mathsf{Gen}^{\mathsf{C}}(\mathsf{param})$: pick $s \leftarrow \mathbb{Z}_q^*$ and set $\mathsf{ck} := h_1 = g^s$, and output $\mathsf{ck}$.
- $\mathsf{Com}_{\mathsf{ck}}(m; r)$: output $c := g^m h_1^r$ and $d := (m, r)$.
- $\mathsf{Open}(c, d)$: output $d := (m, r)$
- $\mathsf{Verify}_{\mathsf{ck}}(c, d)$: return $\mathsf{valid}$ if and only if $c = g^m h_1^r$

Pedersen commitment is also additively homomorphic, i.e.

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

### 4.4 Distributed key generation

Distributed key generation (DKG) is a fundamental building block of the voting process in our proposed treasury system. To ensure robustness, the elected voting committee members invoke the distributed key generation protocol to setup the voting public key such that .

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

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 $\mathsf{HE} = (\mathsf{Gen}^{\mathsf{E}}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Scale}, \mathsf{Dec})$ 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 KeyGen request from at least $t$ key holders $\mathsf{K}_i$, then it generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}^{\mathsf{E}}(\mathsf{param})$.
- Upon receiving input $(\textsc{ReadPK}, \mathsf{sid})$ from party $P \in \mathcal{K} \cup \mathcal{U}$, if $\mathsf{pk}$ 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. 4: Threshold homomorphic vector encryption functionality $\mathcal{F}_{\text{THVE}}^{t,k}$

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 $\mathsf{C}_i$ first generates a public/private key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}^{\mathsf{E}}(\mathsf{param})$. Each $\mathsf{C}_i$ then posts $\mathsf{pk}_i$ to the blockchain and use $(t+1, n)$-threshold *verifiable secret sharing* (VSS) to share $\mathsf{sk}_i$ 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.* [39]. In a nutshell, the protocol lets the committee members $\mathsf{C}_i$ first posts a "commitment" of $\mathsf{pk}_i$. After sharing the corresponding $\mathsf{sk}_i$ via $(t+1, n)$-threhold VSS, the committee members $\mathsf{C}_i$ then reveals $\mathsf{pk}_i$. We will use the blockchain to realises the broadcast channel and peer-to-peer channels.

## 4.5 Unit vector ZK proof with logarithmic size communication

Let $\mathbf{e}_i^{(n)} = (e_{i,0}, \ldots, e_{i,n-1})$ denote a unit vector 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-Pederson proofs [41] 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 to the length of the unit vector, and thus the communication overhead is quit 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 [42], 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. 5.

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 [42]. Note that in the 3rd move of such a Sigma protocol, the prover reveals a degree-1 polynomial of the committed message.

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

Denote $z_{\ell,1} := i_\ell x + r_\ell$, $\ell \in [\log n]$ as the corresponding degree-1 polynomials, where $r_\ell$ are chosen by the prover and $x$ is chosen by the verifier. By linearity, we can also define $z_{\ell,0} := x - z_{\ell,1} = (1 - i_\ell)x - r_\ell$, $\ell \in [\log n]$. According to the algorithm described in Fig.5, 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 $E := \sum_{j=0}^{n-1} (E_j)^{y^j} \cdot \prod_{k=0}^{\log n - 1} (D_k)^{x^k}$ encrypts 0, where $D_\ell := \mathsf{Enc}_{\mathsf{pk}}(\sum_{j=0}^{n-1} (p_{j,k} \cdot y^j); R_\ell)$, $\ell \in [0, \log n - 1]$ with fresh randomness $R_\ell \in \mathbb{Z}_p$. The construction is depicted in Fig. 6, and it consists of 5 moves. Both the prover and the verifier shares a common reference string (CRS), which is a Pedersen commitment key that can be generated using random oracle. The prover first commits to each bits of the binary representation of $i$, and the commitments are denoted as $I_\ell$, $\ell \in [\log n]$. Subsequently, it produces $B_\ell, A_\ell$ as the first move of the Sigma protocol in Section 2.3 of [42] showing $I_\ell$ commits to 0 or 1. Jumping ahead, later the prover will receive a challenge $x \leftarrow \{0,1\}^\lambda$, and it then computes the third move of the Sigma protocols by producing $\{z_\ell, w_\ell, v_\ell\}_{\ell=1}^{\log n}$. To enable batch verification, before that, the prover is given another challenge $y \leftarrow \{0,1\}^\lambda$ in the second move. The prover 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 $I_\ell$ commits to 0 or 1.

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

In the second part, the verifier checks if

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

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

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

**Protocol:**

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

**Verification:**

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

Fig. 6: Unit vector ZK argument

**Theorem 1.** *The protocol described in Fig. 6 is a 5-move public coin special honest verifier zero-knowledge arguement of knowledge of* $\mathbf{e}_i^{(n)}$ *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.* 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

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

For soundness, first of all, the Sigma protocols for commitments of $i_\ell$, $\ell \in [\log n]$ is specially sound, i.e., given two transactions with the same $\{I_\ell, B_\ell, A_\ell\}_{\ell=1}^{\log n}$ and two different $x$ and $\{z_\ell, w_\ell, v_\ell\}_{\ell=1}^{\log n}$, there exists a PPT extractor that can output the corresponding witness $i_\ell \in \{0, 1\}$. The protocol builds and evaluates a degree-$\log n$ polynomial $p_j(x)$. By Schwartz-Zippel lemma, we have $e_{i,j} = \prod_{\ell=1}^{\log n} i_{\ell,j_\ell}$ with overwhelming probability.

## 5 The proposed treasury system

### 5.1 System overview

A treasury period consists of three main epochs: pre-voting epoch, voting epoch, and post-voting epoch. As shown in Figure 7, the pre-voting epoch includes two concurrent stages: project proposing stage and voter/expert registration stage. In the project proposing stage, the users can submit project proposals, asking for treasury funds. Meanwhile, the interested stake holders can register themselves as either voter or expert to participate in the decision making process by depositing certain amount of the underlying cryptocurrency. The voters'/experts' voting powers are proportional to their deposited amount. Analogously, the voters' rewards are proportional to their deposited amount; whereas, the experts' rewards are proportional to their 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 and experts. The probability each voter/expert will be selected is also proportional to their deposit amount. 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 to the projects. For each project, the voters can delegate to different expert. At the post-voting epoch, the voting committee members jointly calculate and announce the tally result on the blockchain. They then sign on the election tally

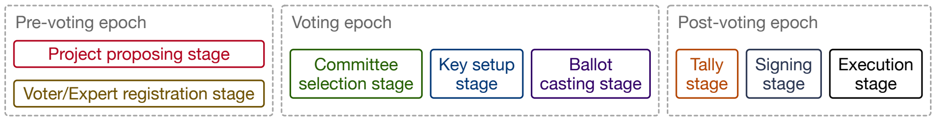

Fig. 7: Treasury system epochs.

result, which will be stored in the main blockchain for a long period. On the contrary, the election transcript will only be kept for a short period to save long-term blockchain storage. Meanwhile, the voting committee members also jointly commit to a random seed, which will be used to select a new voting committee in the next treasury period. Finally, in the execution stage, the winning projects are funded, and the voters, experts, voting committee members are rewarded accordingly.

## 5.2 Treasury funding sources

Funding, perhaps the most crucial ingredient in a decentralised community-controlled decision-making system, must not only be regular, but also be sourced from decentralised means. That is, the source of funding for decentralise treasury system should not introduce centralisation into the system. To this end, desirable properties from the funding sources are

- Sustainable
- Decentralised
- Secure

We note that although it is impossible for all potential funding sources to meet these criteria, a combination of some of these potential sources still meet the set out requirement. Therefore, we propose 3 major sources of funding for the treasury system.

- Taxation from Miners' reward
- Minting new coins
- Donations or Charity

## 5.3 Voter/Expert registration

In order to register to be a voter, a user (or a set of users) need(s) to deposit at least $\alpha_{\min}$ amount of coins. More precisely, the user submit a special *registration transaction* in forms of

$$\mathsf{Tx}\Big(\textsc{Voter-Reg}, \mathsf{treasury\text{-}ID}, \{(\mathsf{coin}_i, \alpha_i)\}_{i=1}^{a}, \overline{\mathsf{pk}}_i\Big) \ ,$$

where $\mathsf{treasury\text{-}ID}$ is the ID of the next treasury event, $\mathsf{coin}_i$ has amount $\alpha_i$ such that $\sum_{i=1}^{a} \alpha_i \geq \alpha_{\min}$, and $\overline{\mathsf{pk}}_i$ is a freshly generated signature verification key for the registered voter. The voter's ID is hash of $\overline{\mathsf{pk}}_i$, denoted as $\mathsf{V}_i := \mathsf{hash}(\overline{\mathsf{pk}}_i)$. Note that this transaction is valid if and only if it is signed by all the owners of the included coins. This also indicates that a registered voter may represent a group of users' opinion, as the deposited money may come from different users.

Similarly, to register as an expert, a user (or a set of users) need(s) to deposit at least $\beta_{\min}$ amount of coins, by submitting a special *registration transaction* in forms of

$$\mathsf{Tx}\Big(\textsc{Expert-Reg}, \mathsf{treasury\text{-}ID}, \{(\mathsf{coin}_j, \beta_j)\}_{j=1}^{b}, \overline{\mathsf{pk}}_j\Big) \ ,$$

where $\mathsf{coin}_j$ has amount $\beta_j$ such that $\sum_{j=1}^{b} \beta_j \geq \beta_{\min}$. The expert's ID is hash of $\overline{\mathsf{pk}}_j$, denoted as $\mathsf{E}_j := \mathsf{hash}(\overline{\mathsf{pk}}_j)$. 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.

### 5.4 Voting committee selection

At the beginning of voting epoch, the voting committee of previous treasury period will jointly decrypt a random seed (denoted as $\mathsf{seed}$) that was committed at the signing stage of the previous post-voting epoch. Once $\mathsf{seed}$ is announced, any registered voter/expert can volunteer to participate in the voting committee selection by submitting a special *registration transaction* in forms of

$$\mathsf{Tx}\Big(\mathrm{VC}, \mathsf{treasury\text{-}ID}, \overline{\mathsf{pk}}_i, \tilde{\mathsf{pk}}, \mathsf{sign}(\mathsf{seed})\Big) \ ,$$

where $\overline{\mathsf{pk}}$ is the signature verification key generated during the voter/expert registration phase, $\tilde{\mathsf{pk}}$ is a freshly generated public key for a pre-defined public key cryptosystem, and $\mathsf{sign}(\mathsf{seed})$ is the signature of $\mathsf{seed}$ under the signing key corresponding to $\overline{\mathsf{pk}}$.

Assume there are $\ell$ volunteers. Let $s_{i_x} := \mathsf{hash}(\overline{\mathsf{pk}}_{i_x}, \mathsf{sign}(\mathsf{seed}))/\mathsf{amount}_{i_x}$, $x \in [\ell]$, where $\mathsf{amount}_{i_x}$ is either $\alpha_{i_x}$ or $\beta_{i_x}$ for voter or expert, respectively. The voting committee member are top $\xi$ (e.g., $\xi = 50$) of the sorted list of $\mathsf{sort}(s_{i_1}, \ldots, s_{i_\ell})$.

### 5.5 Decision making

For each project, the voters/experts need to submit an independent ballot. More precisely, the voting committee members, the voters, and the experts follow the protocol description in Section 5.7. If the voting committee member cheats, he/she will lose all the deposited money.

Proposals are approved for funding in the current treasury period via fuzzy threshold voting. After the voting, the proposal score is the number of $\mathsf{yes}$ votes minus the number of $\mathsf{no}$ votes. Proposals that got at least 10% (of all votes) of the positive difference are acceptable candidates, and all the rest are discarded. Acceptable candidates are ranked according to their score, and the most popular proposal is funded according to the requested amount of money, then the next acceptable candidate, and so on, till the limit is reached.

### 5.6 Post-voting execution

20% of the treasury fund will be used to reward the voting committee members, voters and experts. The voting committee members will receive a fix amount of reward, denoted as $\zeta_1$. The voter $\mathsf{V}_i$ will receive reward that is proportional to his/her deposited amount, denoted as $\zeta_2 \cdot \alpha_i$. The expert $\mathsf{E}_j$ will receive reward that is proportional to his/her received delegations, denoted as $\zeta_3 \cdot D_j$.

In addition, each voting committee members $\mathsf{C}_\ell$, $\ell \in [k]$ will pick a random group element $R_\ell \leftarrow \mathbb{G}$ and post the encryption of it, $C_\ell \leftarrow \mathsf{Enc}_{\mathsf{pk}}(R_\ell)$ to the blockchain. $C := \prod_{\ell=1}^{k} C_\ell$ is defined as the committed/encrypted seed for the next treasury period.

The voting committee members then jointly sign the treasury decision as well as the seed commitment $C$. The signatures will be kept on the blockchain for long-term storage. Finally, the projects will be funded accordingly.

### 5.7 The proposed voting scheme

In this section, we will describe our proxy-voting schemes with pre-defined experts. The formal protocol description is depicted in Figure 8.

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

We define encode as a function that accepts two arguments Voter-ID and one of three options Yes, No, or Abstain.

When the input is from an expert, encode returns the unit vector 100 when choice is Yes, 010 for No and 001 when the input from the expert is Abstain.

However, for other voters, encode returns a unit vector of length m+3, where m is the number of experts. Where m is the number of experts and encode returns a unit vector in a position between 1 and m where a voter delegates their vote. Where voters vote directly by themselves encode returns output similar to the outputs of the encode function when called by an expert.

Formally,

$$Encode(x) = \begin{cases} 100, & \text{if } x = YES \\ 010, & \text{if } x = NO \\ 001, & \text{if } x = ABSTAIN \\ 000, & \text{otherwise} \end{cases}$$

Formally, we define

$$Decode(x) = \begin{cases} YES, & \text{if } x = 100 \\ NO, & \text{if } x = 010 \\ ABSTAIN, & \text{if } x = 001 \\ \perp, & \text{if } x = 000 \end{cases}$$

**Preparation phase:**

- Upon receiving $(\textsc{Init}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the committee $\mathsf{C}_j$, $j \in [k]$ sends $(\textsc{KeyGen}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$,

**Ballot casting phase:**

- Upon receiving $(\textsc{Vote}, \mathsf{sid}, v_i, \beta_i)$ from the environment $\mathcal{Z}$, the expert $\mathsf{E}_i$, $i \in [m]$ does the following:
  - Send $(\textsc{ReadPK}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$, and receive $(\textsc{PublicKey}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{F}_{\mathrm{THVE}}^{t,k}$.
  - Set the unit vector $\mathbf{e}^{(3)} \leftarrow \mathsf{encode}^{\mathsf{E}}(v_i)$. Send $(\textsc{Encrypt}, \mathsf{sid}, \mathbf{e}^{(3)}, \eta)$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ and receive $(\textsc{Ciphertext}, \mathsf{sid}, \mathbf{c}^{(3)})$ from $\mathcal{F}_{\mathrm{THVE}}^{t,k}$.
  - Send $(\textsc{Post}, \mathsf{sid}, \mathbf{c_i}^{(3)}, \beta_i)$ to $\mathcal{F}_{\mathrm{BC}}$.
- Upon receiving $(\textsc{Cast}, \mathsf{sid}, v_j, \alpha_j)$ from the environment $\mathcal{Z}$, the voter $\mathsf{V}_j$, $j \in [n]$ does the following:
  - Send $(\textsc{ReadPK}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$, and receive $(\textsc{PublicKey}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{F}_{\mathrm{THVE}}^{t,k}$.
  - Set the unit vector $\mathbf{e}^{(m+3)} \leftarrow \mathsf{encode}^{\mathsf{V}}(v_j)$. Send $(\textsc{Encrypt}, \mathsf{sid}, \mathbf{e}^{(m+3)})$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ and receive $(\textsc{Ciphertext}, \mathsf{sid}, \mathbf{u}^{(m+3)})$ from $\mathcal{F}_{\mathrm{THVE}}^{t,k}$.
  - Send $(\textsc{Post}, \mathsf{sid}, \mathbf{u_j}^{(m+3)}, \alpha_j)$ to $\mathcal{F}_{\mathrm{BC}}$.

**Tally phase:**

- Upon receiving $(\textsc{DelCal}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the committee $\mathsf{C}_t$, $t \in [k]$ does:
  - Send $(\textsc{ReadBB}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{BC}}$, and receive $(\textsc{ReadBB}, \mathsf{sid}, \mathsf{data})$ from $\mathcal{F}_{\mathrm{BC}}$.
  - Fetch the ballots $\{(\mathbf{c_i}^{(3)}, \beta_i)\}_{i\in[m]}$ and $\{(\mathbf{u_j}^{(m+3)}, \alpha_j)\}_{j\in[n]}$ from $\mathsf{data}$,
  - For $i \in [m]$, send $(\textsc{Check}, \mathsf{sid}, \mathbf{c_i}^{(3)})$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$; for $j \in [n]$, send $(\textsc{Check}, \mathsf{sid}, \mathbf{u_j}^{(m+3)})$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$; Remove the ballots if the $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ response is not $\mathsf{valid}$.
  - 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 $(\textsc{Scale}, \mathsf{sid}, a_{j,\ell}, \alpha_j)$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ and receive $(\textsc{Scale}, \mathsf{sid}, z_{i,\ell})$ from $\mathcal{F}_{\mathrm{THVE}}^{t,k}$.
  - For $i \in [0, m-1]$, send $(\textsc{Add}, \mathsf{sid}, (z_{1,i}, \ldots, z_{n,i}))$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ and receive $(\textsc{Sum}, \mathsf{sid}, s_i)$ from $\mathcal{F}_{\mathrm{THVE}}^{t,k}$;
  - For $i \in [0, m-1]$, send $(\textsc{Decrypt}, \mathsf{sid}, s_i)$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$.
- Upon receiving $(\textsc{Tally}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the committee $\mathsf{C}_t$, $t \in [k]$ does:
  - Send $(\textsc{ReadDec}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$, and receive $(\textsc{Plaintext}, \mathsf{sid}, \mathsf{plaintext})$ from $\mathcal{F}_{\mathrm{THVE}}^{t,k}$.
  - Fetch $\{(s_i, w_i)\}_{i\in[m]}$ from $\mathsf{plaintext}$.
  - For $i \in [0, m-1]$, $\ell \in [0, 2]$, send $(\textsc{Scale}, \mathsf{sid}, c_{i,\ell}, w_i + \beta_i)$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ and receive $(\textsc{Scale}, \mathsf{sid}, d_{i,\ell})$ from $\mathcal{F}_{\mathrm{THVE}}^{t,k}$.
  - For $\ell \in [0, 2]$, send $(\textsc{Add}, \mathsf{sid}, (d_{0,\ell}, \ldots, d_{m-1,\ell}, b_{1,\ell}, \ldots, b_{n,\ell}))$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ and receive $(\textsc{Sum}, \mathsf{sid}, x_\ell)$ from $\mathcal{F}_{\mathrm{THVE}}^{t,k}$;
  - For $\ell \in [0, 2]$, send $(\textsc{Decrypt}, \mathsf{sid}, x_\ell)$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$.
- Upon receiving $(\textsc{ReadTally}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the party $P$ does the following:
  - Send $(\textsc{ReadDec}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{THVE}}^{t,k}$, and receive $(\textsc{Plaintext}, \mathsf{sid}, \mathsf{plaintext})$ from $\mathcal{F}_{\mathrm{THVE}}^{t,k}$.
  - Fetch $\{(x_\ell, y_i)\}_{i\in[0,2]}$ from $\mathsf{plaintext}$, and return $(\textsc{ReadTallyReturn}, \mathsf{sid}, (y_0, y_1, y_2))$ to the environment $\mathcal{Z}$.

Fig. 8: The proxy voting protocol $\Pi_{\textsc{Vote}}^{t,k,m,n}$ in $\{\mathcal{F}_{\mathrm{BC}}, \mathcal{F}_{\mathrm{THVE}}^{t,k}\}$-hybrid model

## 6 Security

**Theorem 2.** *Let $k, n, m = \mathrm{poly}(\lambda)$ and $t > k/2$. Protocol $\Pi_{\text{VOTE}}^{t,k,n,m}$ described in Fig. 8 UC-realizes $\mathcal{F}_{\text{VOTE}}$ in the $\{\mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{THVE}}^{t,k}\}$-hybrid world against static corruption.*

*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 $\mathsf{EXEC}_{\Pi_{\text{VOTE}}^{t,k,n,m},\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\text{BC}},\mathcal{F}_{\text{THVE}}^{t,k}}$ 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 $\{\mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{THVE}}^{t,k}\}$-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 $\mathsf{EXEC}_{\mathcal{F}_{\text{VOTE}},\mathcal{S},\mathcal{Z}}$ where the parties interact with functionality $\mathcal{F}_{\text{VOTE}}$ 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 $\mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{THVE}}^{t,k}$. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

In the preparation phase:

- Upon receiving $(\textsc{InitNotify}, \mathsf{sid}, \mathsf{C}_j)$ 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 $\mathsf{C}_j$, following the protocol $\Pi_{\text{VOTE}}^{t,k,n,m}$ as if $\mathsf{C}_j$ receives $(\textsc{Init}, \mathsf{sid})$ from the environment $\mathcal{Z}$.

In the ballot casting phase:

- Upon receiving $(\textsc{VoteNotify}, \mathsf{sid}, \mathsf{E}_i, \beta_i)$ 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 $(\textsc{Encrypt}, \mathsf{sid}, 0)$ to $\mathcal{F}_{\text{THVE}}^{t,k}$ and receive $(\textsc{Ciphertext}, \mathsf{sid}, c_{i,\ell})$ 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 $(\textsc{Post}, \mathsf{sid}, \mathbf{c_i}^{(3)}, \beta_i)$ to $\mathcal{F}_{\text{BC}}$.
- Upon receiving $(\textsc{CastNotify}, \mathsf{sid}, \mathsf{V}_j, \alpha_j)$ 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 $(\textsc{Encrypt}, \mathsf{sid}, 0)$ to $\mathcal{F}_{\text{THVE}}^{t,k}$ and receive $(\textsc{Ciphertext}, \mathsf{sid}, u_{j,\ell})$ 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 $(\textsc{Post}, \mathsf{sid}, \mathbf{u_j}^{(m+3)}, \alpha_j)$ to $\mathcal{F}_{\text{BC}}$.
- Once the simulated $\mathcal{F}_{\text{BC}}$ receives $(\textsc{Post}, \mathsf{sid}, \mathbf{c_i}^{(3)}, \beta_i)$ 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 $v_i \leftarrow \mathsf{encode}^{-1}((e_{i,0}, \ldots, e_{i,2}))$ and sends $(\textsc{Vote}, \mathsf{sid}, v_i, \beta_i)$ to $\mathcal{F}_{\text{VOTE}}^{t,k}$.

– Once the simulated $\mathcal{F}_{\mathrm{BC}}$ receives $(\textsc{Post}, \mathsf{sid}, \mathbf{u_j}^{(m+3)}, \alpha_j)$ from a corrupted voter $\mathsf{V}_j \in \mathcal{V}_{\mathsf{cor}}$, the simulator $\mathcal{S}$ checks the internal state of $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ for recorded $\{(u_{j,\ell}, e_{j,\ell})\}_{\ell \in [0,m+2]}$ from $\mathsf{ciphertext}$. It then computes $v_j \leftarrow \mathsf{encode}^{-1}((e_{j,0}, \ldots, e_{j,m+2}))$ and sends $(\textsc{Cast}, \mathsf{sid}, v_j, \alpha_j)$ to $\mathcal{F}_{\textsc{Vote}}^{t,k}$.

In the tally phase:

– Upon receiving $(\textsc{DelCalNotify}, \mathsf{sid}, \mathsf{C}_j)$ from the external $\mathcal{F}_{\textsc{Vote}}^{t,k}$ for an honest trustee $\mathsf{C}_j \in \mathcal{C} \setminus \mathcal{C}_{\mathsf{cor}}$, the simulator $\mathcal{S}$ acts as $\mathsf{C}_j$, following the protocol $\Pi_{\textsc{Vote}}^{t,k,n,m}$ as if $\mathsf{C}_j$ receives $(\textsc{DelCal}, \mathsf{sid})$ from $\mathcal{Z}$.
– Upon receiving $(\textsc{TallyNotify}, \mathsf{sid}, \mathsf{C}_j)$ from the external $\mathcal{F}_{\textsc{Vote}}^{t,k}$ for an honest trustee $\mathsf{C}_j \in \mathcal{C} \setminus \mathcal{C}_{\mathsf{cor}}$, the simulator $\mathcal{S}$ acts as $\mathsf{C}_j$, following the protocol $\Pi_{\textsc{Vote}}^{t,k,n,m}$ as if $\mathsf{C}_j$ receives $(\textsc{Tally}, \mathsf{sid})$ from $\mathcal{Z}$.
– Upon receiving $(\textsc{Leak}, \mathsf{sid}, \tau)$ from the external $\mathcal{F}_{\textsc{Vote}}^{t,k}$, the simulator $\mathcal{S}$ parses $\tau$ as $(\tau_0, \tau_1, \tau_2)$ and acts as the simulated $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ to send $(\textsc{DecLeak}, \mathsf{sid}, x_\ell, \tau_\ell)$, $\ell \in [0,2]$, where $x_\ell$ are the tally ciphertexts received by $\mathcal{F}_{\mathrm{THVE}}^{t,k}$ for final decryption.

## 7 Implementation and Performance

### 7.1 Elliptic Curve Over $\mathbb{F}_p$

The implementation of our scheme is based on elliptic curve groups for efficiency. Let $\sigma := (p, a, b, g, q, \zeta)$ be the elliptic curve domain parameters over $\mathbb{F}_p$, consisting of a prime $p$ specifying the finite field $\mathbb{F}_p$, two elements $a, b \in \mathbb{F}_p$ specifying an elliptic curve $E(\mathbb{F}_p)$ defined by $E : y^2 \equiv x^3 + ax + b \pmod{p}$, a base point $g = (x_g, y_g)$ on $E(\mathbb{F}_p)$, a prime $q$ which is the order of $g$, and an integer $\zeta$ which is the cofactor $\zeta = \#E(\mathbb{F}_p)/q$. We denote the cyclic group generated by $g$ by $\mathbb{G}$, and it is assumed that the DDH assumption holds over $\mathbb{G}$, that is for all p.p.t. adversary $\mathcal{A}$:

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{DDH}}(\mathcal{A}) = \left| \Pr\left[ \begin{matrix} x, y \leftarrow \mathbb{Z}_q; b \leftarrow \{0,1\}; h_0 = g^{xy}; \\ h_1 \leftarrow \mathbb{G} : \mathcal{A}(g, g^x, g^y, h_b) = b \end{matrix} \right] - \frac{1}{2} \right| \leq \epsilon(\lambda) \ ,$$

where $\epsilon(\cdot)$ is a negligible function.

### 7.2 Performance metrics

The Treasury architecture relies on underling cryptocurrency blockchain and uses its properties. The main point of interest for the Treasury performance testing is the proposed voting protocol, as it provides main computation workload for running nodes.

So, in performance testing it is needed to get evaluation of computation complexity as well as and space complexity for message transmission over peer-to-peer network of

– generation of the election public key;
– ballot validation using NIZK proof of its correctness and getting tally results.

Thus, we are interested in computation time required by voting committee members, size of messages for election key generation, computation workload for a node that fully verifies election results and amount of information needed to be sent over peer-to-peer network during one Treasury cycle (epoch).

## 7.3 Test bench environment

For evaluating protocol performance it was developed a reference implementation using Scala programming language. The source code is based on Scala 2.12.3. Elliptic curve math was implemented on BouncyCastle (ver.1.58) Java library with the standard secp-256k1 curve to run the protocol. Finite field transformations were made using BigInteger class from Java Core. Subprotocols of the developed system were implemented exactly as they are described in the paper without any protocol-level optimizations. Tests were done on the machine with the following configuration:

We used the followng archtecture and environment for benchmarking of the voting protocol:

**CPU:** Intel Core i7-6500U CPU @ 2.50GHz
**RAM:** 16 GB
**OS:** Linux Ubuntu 16.04 64 bit
**Scala:** scalaVersion := "2.12.3"
**Java:** OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
**EC Math library:** org.bouncycastle "1.58"

All performance benchmarks are obtained for single-threaded version of the implementation. Providing multithreaded execution and native code implementation, as well as protocol-level optimizations, allow further improvement of running time.

## 7.4 Performance results

**Generation of the election public key.** We benchmarked key generation protocol running time for different number of voting committee members: from 10 to 100 (so high numbers might be required to guarantee honest committee majority on member random selection among large amount of voters). Election 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 Figure 9.

Besides it, there were estimated amount of data needed to be transmitted over the peer-to-peer network to complete the protocol, in dependence of committee

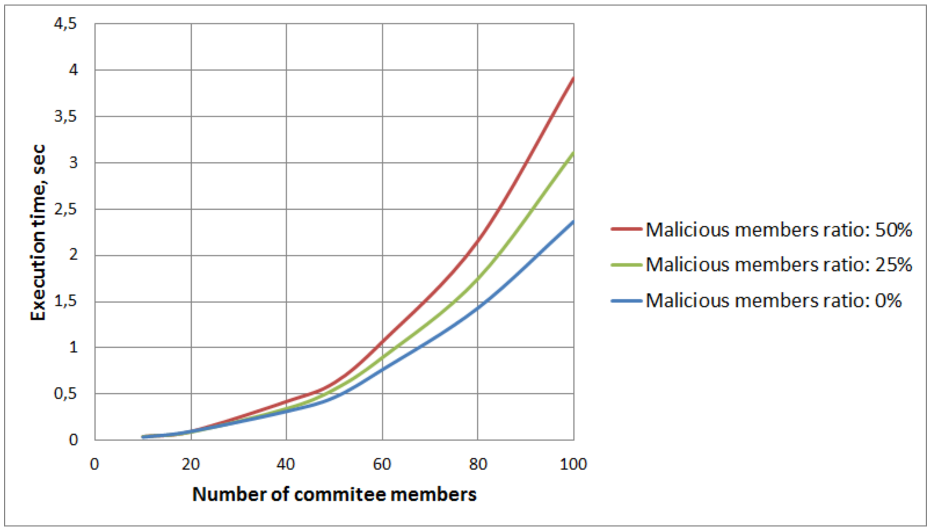

Fig. 9: DKG protocol execution time depending on the number of commitee members

members number and malicious members ratio. Results are given in Figure 10 (we recall here that even controlling 50% of the committee, an attacker can break confidentiality of voters' ballots, but not their integrity or tally result).

It also should be noted that for election public key generation we have implemented a modified version of the protocol given in Figure **??**. It provides better performance and space load, has the same properties, but there are no published proofs of its UC-properties. They will be provided in the extended version of our paper.

**Ballots and tally.** Ballot generation is done once by the voter and takes less than 1 second up to 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.

Dependence of tally execution time, combined with each ballot NIZK validation, on number of voters, is given in Figure 11 (the number of experts is constant and equals 50 for all benchmarked number of voters).

Dependence of the ballot size and NIZK proofs of its correctness (both for voters and experts) on total number of experts are given in Figure 12.

Thus, benchmarking results shows quite acceptable system workload. Besides it, there are also further effective ways for performance improvement pro-

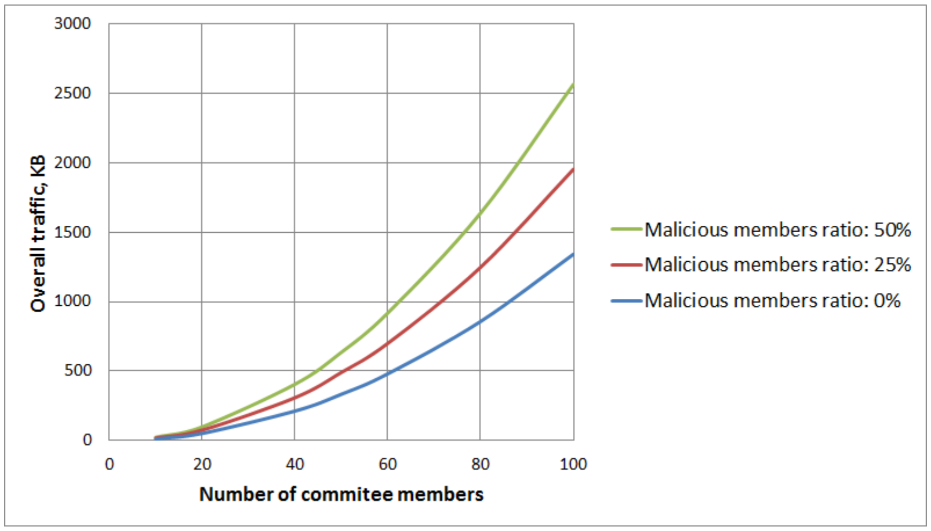

Fig. 10: Total size of the DKG protocol messages to be sent over the peer-to-peer network depending on the number of commitee members

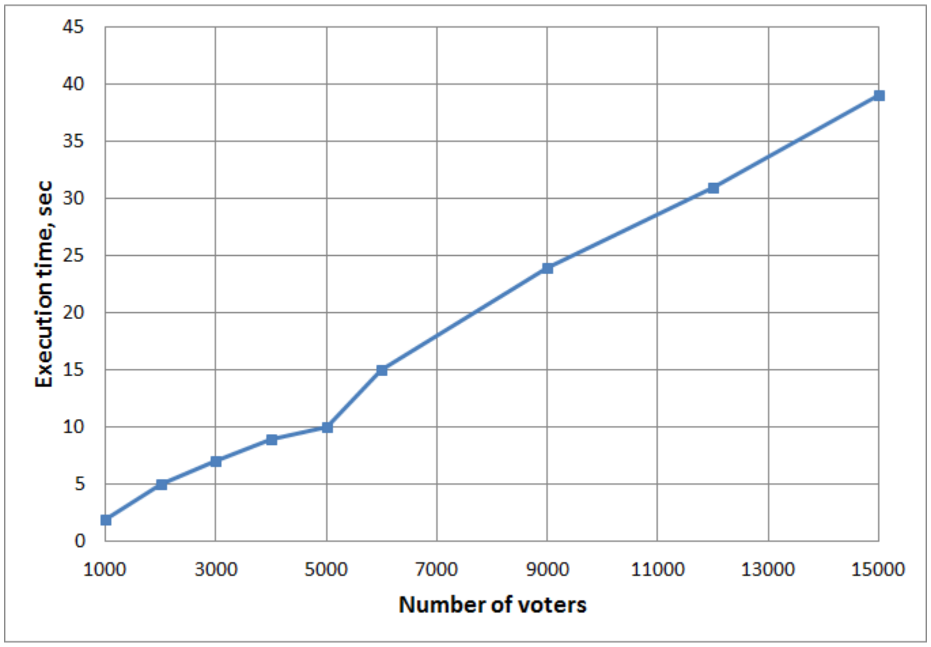

Fig. 11: Dependence of tally execution time together with ballots NIZK validation on number of voters

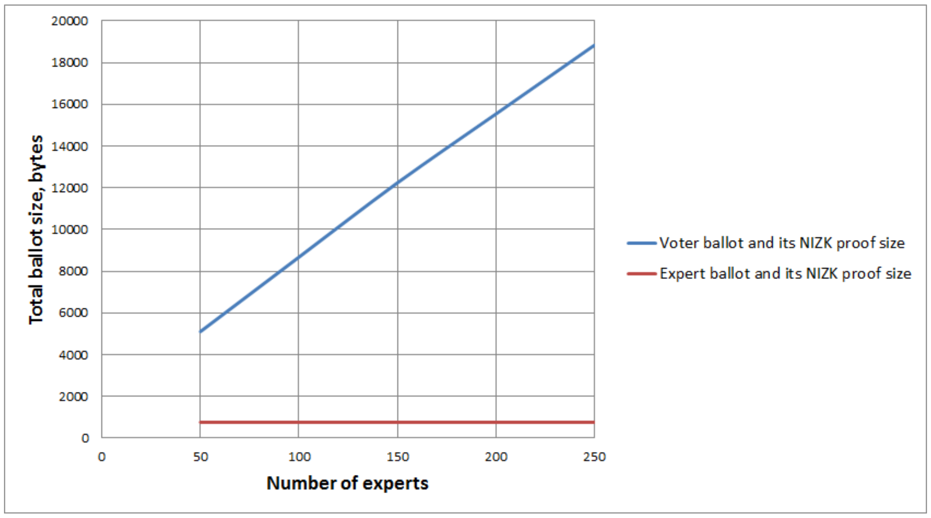

Fig. 12: Ballot size and NIZK proofs of its correctness depending on number of experts

viding native code implementation instead of VM execution, combined with multithreading.

## 8 Acknowledgements

We thank Dmytro Kaidalov and Andrii Nastenko from IOHK for the full prototype implementation of the proxy voting protocol, all its subprotocols and algorithms, distributed key generation and their performance benchmarks.

## References

1. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
2. "Cryptocurrency market capitalizations." https://coinmarketcap.com/.
3. "Blockchain tracker: Governments trust in blockchain." http://fintechranking.com/2017/02/04/blockchain-tracker-governments-trust-in-blockchain/.
4. E. Duffield and D. Diaz, "Dash: A privacycentric cryptocurrency." https://www.dash.org/wp-content/uploads/2015/04/Dash-WhitepaperV1.pdf.
5. D. Kaidalov, A. Nastenko, *et al.*, "Dash governance system: Analysis and suggestions for improvements." https://iohk.io/research/papers/#NSJ554WR.
6. A. Blais and L. Massicotte, "Electoral systems," *Comparing democracies*, vol. 2, pp. 40–69, 2002.
7. U. Endriss, "Voting procedures and their properties." https://formal.iti.kit.edu/teaching/FormSys2SoSe2016/02socialchoiceB.pdf, 2016. Online; date last accessed: 2017-10-26.
8. D. Cary, "Estimating the margin of victory for instant-runoff voting.," in *EVT/WOTE*, 2011.
9. P. Emerson, "The original borda count and partial voting," *Social Choice and Welfare*, pp. 1–6, 2013.
10. N. Tideman, "The single transferable vote," *The Journal of Economic Perspectives*, vol. 9, no. 1, pp. 27–38, 1995.
11. 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.
12. 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.
13. P. C. Fishburn and S. J. Brams, "Paradoxes of preferential voting," *Mathematics Magazine*, vol. 56, no. 4, pp. 207–214, 1983.
14. S. J. Brams and P. C. Fishburn, "Approval voting," *American Political Science Review*, vol. 72, no. 3, pp. 831–847, 1978.
15. "Approval voting." https://electology.org/approval-voting. Online; date last accessed: 2017-10-26.
16. E. Duffield and D. Diaz, "Dash: A privacy-centric crypto-currency," 2014.
17. D. Harrison, "Decentralized autonomous organizations." http://www.allenovery.com/SiteCollectionDocuments/Article%20Decentralized%20Autonomous%20Organizations.pdf, May 16 2016. Online; date last accessed: 2017-10-21.
18. L. F. Molina, "Fermat, the internet of people and the person to person economy." https://hackernoon.com/fermat-the-internet-of-people-and-the-person-to-person-economy-ce933865a0b0, 2016. Online; date last accessed: 2017-10-26.
19. D. Kaidalov, L. Kovalchuk, *et al.*, "A proposal for an ethereum classic treasury system." https://iohk.io/research/papers/#AJSEAT7K.
20. B. Ford, "Delegative democracy," *Manuscript*, 2002.
21. 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.
22. D. Lomax, "Beyond politics, an introduction," January 1, 2003. Online; date last accessed: 2017-10-21.

23. 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.
24. 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.
25. J. Green-Armytage, “Direct voting and proxy voting,” *Constitutional Political Economy*, vol. 26, no. 2, pp. 190–220, 2015.
26. J. Ito, “Emergent democracy,” *Retrieved November*, vol. 17, p. 2007, 2003.
27. S. Hardt and L. C. Lopes, “Google votes: A liquid democracy experiment on a corporate social network,” 2015.
28. Proxy.me, “Voteflow.” http://www.proxyfor.me/help, 2015. Online; date last accessed: 2017-10-21.
29. LiquidFeedback, “LiquidFeedback official website.” http://liquidfeedback.org. Online; date last accessed: 2017-10-21.
30. Adhocracy, “Adhocracy official website.” https://adhocracy.de. Online; date last accessed: 2017-10-21.
31. J. Degrave, “Getopinionated.” https://github.com/getopinionated/getopinionated. GitHub repository; date last accessed: 2017-10-21.
32. Democracy Earth, “The social smart contract. an open source white paper..” http://democracy.earth, September 1, 2017. Online; date last accessed: 2017-10-21.
33. S. McCarthy, “Wasa2il.” https://github.com/smari/wasa2il, 2016. GitHub repository; date last accessed: 2017-10-21.
34. nVotes, “3 crypto schemes for liquid democracy (iii).” https://nvotes.com/3-crypto-schemes-liquid-democracy-iii/, July 19 2017. Online; date last accessed: 2017-10-21.
35. B. Zhang and H.-S. Zhou, “Statement voting.” Cryptology ePrint Archive, Report 2017/616, 2017. http://eprint.iacr.org/2017/616.
36. 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.
37. 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.
38. J. T. Schwartz, “Fast probabilistic algorithms for verification of polynomial identities,” *J. ACM*, vol. 27, pp. 701–717, Oct. 1980.
39. 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.
40. D. Wikström, “Universally composable DKG with linear number of exponentiations,” in *SCN 2004*, pp. 263–277, Springer Berlin Heidelberg, 2004.
41. D. Chaum and T. P. Pedersen, “Wallet databases with observers,” in *CRYPTO 1992 Proceedings*, vol. 740 of *LNCS*, pp. 89–105, Springer, 1993.
42. 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.