Universally Composable On-Chain Quadratic Voting for Liquid Democracy

Lyudmila Kovalchuk, Bingsheng Zhang, Andrii Nastenko, Zeyuan Yin, Roman Oliynykov, Mariia Rodinko

***Abstract*—Decentralized governance plays a critical role in blockchain communities, allowing stakeholders to shape the evolution of platforms such as Cardano, Gitcoin, Aragon, and MakerDAO through distributed voting on proposed projects to support the most beneficial ones. In this context, numerous voting protocols for decentralized decision-making have been developed, enabling secure and verifiable voting for individual projects (proposals). However, these protocols are not designed to support more advanced models such as quadratic voting (QV), where the voting power, defined as the square root of a voter's stake, must be distributed among the projects selected by the voter. Simply running multiple instances of a single-choice voting scheme in parallel is insufficient, as it cannot enforce the correct splitting of voting power. To address this, we propose an efficient blockchain-based voting protocol that supports liquid democracy under the QV model, while ensuring voter privacy, fairness, and verifiability of voting results. In our scheme, voters can delegate their votes to trusted representatives (delegates), while having the ability to distribute their voting power across selected projects. We model our protocol in the Universal Composability framework [1] and formally prove its UC-security under the Decisional Diffie–Hellman (DDH) assumption. To evaluate the performance of our protocol, we developed a prototype implementation [2] and conducted performance testing. The results show that the size and processing time of a delegate's ballot scale linearly with the number of projects, while a voter's ballot scales linearly with both the number of projects and the number of available delegation options. In a representative setting with 64 voters, 128 delegates, and 128 projects, the overall traffic is approximately 1.4 MB per voted project, confirming the practicality of our protocol for modern blockchain-based governance systems.**

***Index Terms*—Blockchain, ZK-proofs, liquid democracy, quadratic voting, on-chain governance.**

I. Introduction

In modern societies, collaborative decision-making is a crucial process that affects the community's overall prosperity and the well-being of each individual. Blockchain communities, in particular, serve as a vivid example where collaborative decision-making plays an extremely important role.

L. Kovalchuk is with G. E. Pukhov Institute for Modelling in Energy Engineering, Ukraine (e-mail: lyudmila.kovalchuk@pimee.ua).

B. Zhang is with IOG Singapore Pte Ltd. and The State Key Laboratory of Blockchain and Data Security, Zhejiang University (e-mail: bingsheng@zju.edu.cn).

A. Nastenko is with Kharkiv National University of Radio Electronics, Ukraine (e-mail: andrii.nastenko@nure.ua).

Z. Yin is with The State Key Laboratory of Blockchain and Data Security, Zhejiang University (e-mail: zeyuanyin@zju.edu.cn).

R. Oliynykov is with V.N. Karazin Kharkiv National University, Ukraine, and Kharkiv National University of Radio Electronics, Ukraine (e-mail: roman.oliynykov@karazin.ua).

M. Rodinko is with V.N. Karazin Kharkiv National University, Ukraine (e-mail: mariia.rodinko@karazin.ua).

Many blockchain platforms and DAOs incorporate governance mechanisms that allow decisions to be made through voting by token holders or members. Notable examples are Cardano, Gitcoin, Aragon, Tezos, Polkadot, MakerDAO, and many others. The core idea behind these systems is to decentralize control over changes and decisions, ideally leading to a more resilient and community-driven ecosystem.

Key aspects of decentralization and governance in modern treasury systems are examined and analyzed in various studies, for example, in [3] using the Edinburgh Decentralization Index (EDI) methodology and in [4] within the Institutional Possibilities Frontier (IPF) framework. Important conclusions that can be drawn from these studies are that blockchain communities should avoid centralizing underlying resources around relevant parties in blockchain systems, and regularly improve their governance systems (by self-funding RnD projects) to mitigate dictatorship and enforce trust in treasury functions.

The democratic approach is widely recognized as an effective method of decentralizing the voting power in decision-making processes through the collaborative participation of all community members. However, direct democracy is not always the most efficient approach within blockchain systems. Making informed decisions requires significant effort from stakeholders to understand the issues at hand and may also require expert knowledge throughout the process. To address this, the concept of liquid democracy has been introduced, offering a balanced approach between direct and representative democracy. This model, also known as delegative democracy, combines the strengths of both approaches while minimizing their weaknesses. It allows stakeholders to either vote directly or delegate their voting power to a trusted expert in the respective field. Such a system enables the use of expert knowledge in decision-making while ensuring broad stakeholder participation and decentralization.

Zhang et al. [5] proposed a treasury system that supports liquid democracy, where stakeholders can either vote independently or delegate their voting power to a trusted expert, while the voting power (i.e., stake) is the same for each proposal a stakeholder decides to select. This system has been implemented and successfully deployed within the Catalyst's community treasury.

The further advancement of collaborative governance was given in the article on prospective voting schemes [6]. Authors examined the general model of quadratic voting (QV) in which the square root can be applied either to the entire

This work was funded in part by Input Output (iog.io)

wallet and/or to the preferential splitting of credits assigned to proposal choices. According to recent studies [7], [8], the QV model with voting power splitting can be considered the most efficient at maximizing the benefit each participant receives from the community's decisions. Importantly, a voter cannot reuse his voting power to support multiple proposed options – it must be divided among them. In this regard, the implementation of QV with voting power splitting in modern DAO systems significantly decentralizes decision-making, thereby being highly beneficial for blockchain communities. The analysis in this paper focuses on the scheme in which the square root is applied to the wallet holdings of the voter, and the voters split allocation of these credits to proposals is linear.

Problem statement. The main challenge is that in the blockchain context, the verifiability, privacy, and fairness of voting are extremely important properties. At the same time, existing blockchain voting schemes that possess these properties are designed to use voting power (usually, a voter's stake) as an atomic value that cannot be distributed among the choices each voter makes. Such schemes are inapplicable for quadratic voting with voting power splitting, so the question is:

> *How to build an efficient blockchain voting scheme that supports liquid democracy under the quadratic voting model with voting power splitting and provides verifiability, privacy, and fairness of voting?*

A. Our contributions

In this work, we aim to construct an efficient, verifiable blockchain voting protocol that implements liquid democracy under the quadratic voting model with guarantees of voting privacy and fairness. In such a scheme, a voter should only be able to distribute his voting power - the square root of his stake - among projects or delegates of his choice, but not reuse it for multiple choices. The main contributions of our work we describe as follows:

- We present a universally verifiable blockchain-based protocol for private and fair distributed voting in liquid democracy under the Quadratic Voting model.
  In our approach, the ballot structure is based on the concept outlined in [5]: a vote is represented as a unit vector (UV) issued by the voter for each selected project. However, a voter, but not a trusted committee, multiplies the UV by a preferred share of its square-rooted public stake (which serves as voting power in our scheme). This is necessary because the voter's stake share must remain private, meaning no one else can perform this multiplication on an encrypted UV. As a result, the voter's ballot now contains both the original UV and the UV multiplied by the stake share.
  To ensure that the voter correctly performs this multiplication, we use a Zero-Knowledge Proof (ZKP), which is a modified version of the Chaum-Pedersen protocol [9] for proving the equality of discrete logarithms. This ZKP proves that the same secret value of a stake share was used as the multiplier for each element of the original UV to produce the multiplied UV, and that the multiplier matches the Pedersen commitment [10] to this stake share (in turn, the correctness of the original UV is proven using the "*Unit vector ZK argument*" from [5]).
  We involve Pedersen commitment in our construction both to hide the stake's share value and to create a Range Proof that ensures this value is non-negative.
  Finally, to guarantee the correct splitting of the publicly known stake into private shares, the voter provides a randomness value corresponding to the sum of the randomness values used to generate the commitments for these stake shares. This allows for easy verification of the correct stake splitting by multiplying all provided commitments and checking whether the result equals the Pedersen commitment of the publicly known stake, given the provided sum of randomness values. In other words, the voter just opens the commitment of the sum of the committed stake shares.
  A similar approach is used when creating a delegate's ballot. The only difference is that instead of splitting its own stake, the delegate splits a virtual stake (identical for all delegates) to express its preferences for selected projects.
  All other structural parts of the protocol remain almost the same as in [5], with the difference that during the vote-counting stage, the trusted committee does not multiply the unit vector by the publicly known stake of the corresponding voter. Instead, it validates and sums the unit vectors that the voters themselves have multiplied by the stake shares.
- We prove that our voting scheme is UC-secure under the DDH assumption by presenting its formal security model via an ideal functionality $\mathcal{F}_{\text{Vote}}^{t,u,m,n}$ within the well-known Universally Composable framework [1]. This functionality interacts with sets of n voters, m delegates, and u voting committee members. Voters can either cast their votes directly or delegate their voting power to designated delegates. The votes cast are weighted by the corresponding shares (portions) of a publicly known stake held by a voter/delegate. The system guarantees termination as long as at least t out of the u committee members behave honestly.
- We provide a prototype implementation [2] to evaluate the performance of our protocol. The benchmark results presented in Section IV show that the size and processing time of the delegate's ballot grow linearly with the number of projects. While the voter's ballot also exhibits linear growth with respect to the number of projects, it additionally scales linearly with the number of delegates due to the support for liquid democracy (details on this are in Section IV).
  When a delegate selects 32 projects, the resulting ballot is 55KB, and its creation/validation takes 146ms/48ms, respectively. When a voter selects 32 projects while having 64 delegation options, the resulting ballot is 351 KB, and its creation/validation takes 587ms/410ms, respectively.

B. Related Work

In the context of the practical implementation of liquid democracy, there are many open problems to explore, as well as many approaches to implementing the corresponding voting schemes. Nejadgholi et al. [11] state that, in addition to ballot privacy, a number of additional properties are important, including knowledge of incoming weight and accountability for delegates, as well as coercion resistance of the voting system. Yin et al. [12] introduce a scalable, coercion-resistant voting scheme based on the fake credentials concept, while Karoukis [13] presents a commitment scheme that enables transparent validation of transfers and reversible delegations of voting power between voters, without sacrificing their privacy, in the context of liquid democracy.

Among other approaches and protocols for implementing decentralized decision-making systems it is worth noting the voter's opinions aggregation with privacy maintained within anonymity sets [14]; the possibility for voters to update their choice during the election process to more flexibly express their opinion without waiting for the next voting round as well as to avoid the so-called "peak-end effect" [15]; the statement voting approach, in which a voter's vote remains indeterminate until the moment of tallying, enabling the implementation of methods like Single Transferable Vote (STV) or liquid democracy, both of which are special cases of statement voting [16]. However, none of the aforementioned approaches, nor many others, seem suitable for implementing liquid democracy in the context of quadratic voting.

Another observation regarding the existing collaborative decision-making schemes in distributed systems is that they are usually implemented as cryptographic protocols based on mechanisms such as Distributed Key Generation (DKG), homomorphic threshold encryption, homomorphic commitment schemes, verifiable secret sharing (VSS), zero-knowledge proofs (ZKP), Verifiable Delay Functions (VDF), and many other modern cryptographic techniques. It should also be noted that distributed voting schemes usually have a set of trusted entities [5], [17]–[19], which allows these schemes to scale for a significant number of voters with a modest number of trustees (often with some redundancy in case some of them fail or are corrupted). Additionally, in the case of blockchain-based voting schemes, voting power is usually tied to the number of tokens a voter holds.

In particular, the Cardano governance system allows each stakeholder to vote on proposed projects using their stake as voting power that is reused across all projects the stakeholder (voter) wishes to support. Although the voter's stake is public, all ballots that contain the actual project choice are issued in encrypted form, and only the final vote count is public. The cryptographic protocol that realizes this voting functionality is described in [5].

To be more specific, Zhang et al. in [5] propose a provably secure protocol for blockchain governance in a liquid democracy. In brief, the protocol is based on counting votes cast as unit vectors (UVs), where each element is encrypted using additively homomorphic encryption, specifically the ElGamal encryption scheme. The only non-zero element of UV represents one of the three options "Yes-No-Absatin" or the identifier of the delegate to whom the choice is being delegated. A trusted committee weighs each ballot with the corresponding stake of the voter who cast the ballot by homomorphically multiplying all encrypted UV elements by the voter's public stake. The final tally is a project-wise sum of all stake-weighted UVs, jointly decrypted by trusted committee members using a shared secret key distributed among them via a threshold secret-sharing scheme. This approach ensures privacy, fairness, and universal verifiability of the voting procedure, however it does not allow voters to distribute their stake across projects, since each UV corresponds to exactly one project, and the voter's stake is the same for weighting each UV cast by the voter.

One of the earliest works addressing secure quadratic voting is [20]. It provides an overview of the main security challenges arising in the secure implementation of a more classical variant of quadratic voting, which we refer to as QV Type 2 [21].

In QV Type 2, stakeholders split their stake among proposals, and during tallying, the sum of the square roots of the corresponding contributions is computed for each proposal, with each contribution corresponding to a single stakeholder. The work combines end-to-end verifiable voting techniques with anonymous payment mechanisms and also discusses alternative refund rules for quadratic voting, such as a lottery-based refund scheme. The contribution of this work is primarily survey-oriented and architectural. It provides a detailed discussion of the security requirements for quadratic voting systems, including verifiability, ballot secrecy, and robustness against false accusations, and outlines cryptographic mechanisms that may be used to achieve these properties, such as homomorphic encryption, threshold decryption schemes, and zero-knowledge proofs. The paper analyzes how these tools can be combined, but does not present a concrete, fully specified cryptographic protocol. In contrast to our work, the paper neither defines an ideal functionality nor provides a formal security proof in the Universal Composability (UC) model. Moreover, the considered protocol is not designed for quadratic voting with split stake across multiple projects, as it focuses on voting between only two alternatives. Finally, unlike our approach, the proposed architecture does not support delegation or liquid democracy and is not suitable for on-chain voting environments.

Among recent studies on QV, the work of [22] is closest to ours in terms of problem formulation. The authors propose *QV-net*, a decentralized self-tallying QV scheme providing maximal ballot secrecy. Self-tallying allows anyone to compute the election outcome once all ballots have been cast, while maximal ballot secrecy ensures that the partial tally of a subset of voters' ballots can only be recovered by the coalition of all remaining voters. The work [22] considers a more classical variant of QV [21], where a stakeholder splits their voting power among proposals. During tallying, the square roots of the corresponding contributions are computed for each proposal (with each contribution corresponding to one stakeholder). The proposed protocol relies on zero-knowledge tools similar to those used in our work, including the distributed key generation protocol, the discrete logarithm

knowledge relation, the discrete logarithm equality relation, the sum-of-squares relation, and range proofs. However, the use of a multi-round sum-of-squares relation requires technically involved constructions and relies on the sum-check protocol [23], introducing additional challenges addressed in that work. While the protocol does not require a trusted third party, it is interactive and involves multiple rounds of communication among participants. Although the authors occasionally mention the possibility of a non-interactive instantiation and note that certain subprotocols could be made non-interactive, these statements lack formal proofs and do not constitute a complete protocol suitable for direct on-chain deployment.

A classical QV setting is also considered in [24], which proposes an end-to-end verifiable QV scheme with everlasting privacy. However, this approach does not support delegation, does not consider an on-chain execution model, and does not provide security guarantees in the UC framework. Voting is performed sequentially: a voter casts a vote for a single proposal using the Camenisch protocol [25] to prove that one exponent is the square of another, and then updates the remaining unused stake. Such a sequential approach is unsuitable for proposal-based on-chain governance systems. Worth mentioning that our work extends this line of research to on-chain liquid democracy with split voting power (VP).

Beyond cryptographic protocol design, QV has also been studied from an economic and information-theoretic perspective. The work of [26] analyzes the behavior of QV in decentralized systems under imperfect information, which is a typical scenario for decentralized governance systems such as DAOs. Although cryptographic privacy is not the main focus of this work, its results highlight important limitations and challenges of QV that must be taken into account when designing privacy-preserving voting protocols. In particular, the fact that QV is not always optimal under imperfect information provides an additional motivation for developing correct, private, and verifiable protocols.

QV and its generalizations have also been explored as mechanisms to reduce power concentration in blockchain-based decision-making [6], where the authors show that QV can improve decentralization but emphasize the need for additional security and privacy mechanisms.

Furthermore, QV has been applied in the context of federated learning for model aggregation [27], demonstrating its broader applicability in privacy-preserving systems, though it does not address the problem of private on-chain voting.

Table I compares our protocol with prior work that proposes concrete cryptographic voting protocols, rather than purely economic or conceptual analyses of quadratic voting.

TABLE I
COMPARISON OF CRYPTOGRAPHIC PROTOCOLS FOR QV

Work	Privacy	Verif.	On-chain	Split VP	Liquid Democracy	UC Security
Pereira & Rønne [24]	+	+	–	–	–	–
QV-net [22]	+	+	–	–	–	–
Our scheme	+	+	+	+	+	+

C. Structure of the Paper

The remainder of this paper is organized as follows. Section II introduces the notations as well as formal definitions of the cryptographic primitives and notions used in this work. In Section III, we present the proposed voting scheme, together with the corresponding security model, protocol description, and formal security analysis within the UC framework. Section IV evaluates the practical performance of the protocol based on a prototype implementation, with a focus on efficiency and scalability. Finally, Section V concludes the paper and outlines directions for future work.

II. PRELIMINARIES

A. Notations

In this paper, we will use the following notations. Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\mathsf{Gen}^{\mathsf{G}}(1^\lambda)$ be the algorithm that takes as input the security parameter $\lambda \in \mathbb{N}$, and outputs the group parameters param, which define a multiplicative cyclic group $\mathbb{G}$ of prime order $q = |\mathbb{G}|$, where $|q| = \lambda$, and a group generator g. We assume the DDH assumption [28] holds with respect to the group $\mathbb{G}$. We abbreviate *probabilistic polynomial time* as PPT.

We denote the set $\{a, a+1, \ldots, b\}$ as $[a, b]$, and $[1, b]$ as $[b]$. When S is a set, $s \leftarrow S$ stands for sampling s uniformly from S. By $\mathbf{a}^{(\ell)}$ we denote a length-ℓ vector $(a_1, \ldots, a_\ell)$. Also, we denote a negligible function by $\mathsf{negl}(\cdot)$ and a polynomially-bounded function by $\mathrm{poly}(\cdot)$.

B. Additively homomorphic encryption

In this work, we use a variant of the ElGamal encryption scheme [29] - the Lifted ElGamal encryption as a well-known additively homomorphic public key cryptosystem consisting of the following algorithms (note that all of them implicitly take as input the same group parameters, $\mathsf{param} \leftarrow \mathsf{Gen}^{\mathsf{G}}(1^\lambda)$):

- $\mathsf{Gen}^{\mathsf{E}}(\mathsf{param})$: pick $\mathsf{sk} \leftarrow \mathbb{Z}_q^*$, set $\mathsf{pk} := g^{\mathsf{sk}}$, and output $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}_{\mathsf{pk}}(m, r)$: output $c := (c_1, c_2) = (g^r, g^m \cdot \mathsf{pk}^r)$.
- $\mathsf{Mul}(c_1, \ldots, c_\ell)$: output $c := (\prod_{i=1}^{\ell} c_{i,1}, \prod_{i=1}^{\ell} c_{i,2})$.
- $\mathsf{Exp}(c, n)$: output $c^n := (c_1^n, c_2^n)$.
- $\mathsf{Dec}_{\mathsf{sk}}(c)$: output $\mathsf{Dlog}(c_2 \cdot c_1^{-\mathsf{sk}})$, where $\mathsf{Dlog}(x)$ is the discrete logarithm of x (note that the message space should be as small as possible since $\mathsf{Dlog}(\cdot)$ is not efficient).

When we apply any of the algorithms $\{\mathsf{Enc}_{\mathsf{pk}}(\cdot), \mathsf{Dec}_{\mathsf{pk}}(\cdot), \mathsf{Add}(\cdot), \mathsf{Exp}(\cdot)\}$ to a vector/s of input parameters, we mean element-wise application of the specified algorithm to the vector/s, such as:

$$\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}^{(\ell)}, \mathbf{r}^{(\ell)}) = (\mathsf{Enc}_{\mathsf{pk}}(m_1, r_1), \ldots, \mathsf{Enc}_{\mathsf{pk}}(m_\ell, r_\ell)).$$

The Lifted ElGamal encryption scheme can be applied to threshold encryption and is IND-CPA secure under the DDH assumption [30]. The additive homomorphic property of this scheme means the following:

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

C. Pedersen commitment

We use Pedersen commitment [10] in our protocol to hide and bind private shares of a voter's public stake. It is perfectly hiding and computationally binding under the DDH assumption and consists of the following algorithms assumed to run in a common context of group parameters, $\mathsf{param} \leftarrow \mathsf{Gen}^{\mathsf{G}}(1^{\lambda})$:

- $\mathsf{Gen}^{\mathsf{C}}(\mathsf{param})$: pick $h \leftarrow \mathbb{G}$, set $\mathsf{ck} = h$ and output ck.
- $\mathsf{Com}_{\mathsf{ck}}(m, r)$: set $d_c := (m, r)$ and output $c := g^m \cdot \mathsf{ck}^r$.
- $\mathsf{Open}(c)$: output d_c.
- $\mathsf{Verify}_{\mathsf{ck}}(c, d)$: return valid if and only if $c = g^m \cdot \mathsf{ck}^r$, where $(m, r) = d$.

Pedersen commitment is also additively homomorphic, i.e.:

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

D. Non-Interactive Zero-Knowledge Proofs/Arguments (NIZK)

In our protocol design, we utilize Sigma protocols with Fiat-Shamir transform [31] in the random oracle model [32] to construct non-interactive zero knowledge proofs/arguments of knowledge. There are three NIZK protocols that we use in our scheme: (i) the non-interactive unit vector proof NIZK_{UV}^{RO}; (ii) the exponentiation and re-encryption correctness proof NIZK_{Exp}^{RO}; (iii) and the range proof NIZK_{RP}^{RO}. For brevity, we omit the RO superscript in the following usages of these notations when the context is clear.

We give formal definitions of Sigma protocols, arguments of knowledge, non-interactive zero-knowledge arguments/proofs of knowledge as well as the descriptions of NIZK_{UV}^{RO}, NIZK_{Exp}^{RO}, and NIZK_{RP}^{RO} in Appendix A. Also, we give the construction of the NIZK_{Exp}^{RO} protocol in Appendix B.

E. Universal composability

We model the security of our voting protocol within the Universal Composability (UC) framework [1], which defines protocol security via the indistinguishability between the environment $\mathcal{Z}$'s views in the real/ideal world executions. In this framework, the intended security properties of a protocol are specified by an ideal functionality – a fully trustworthy third party that performs the protocol's task in the ideal world. It explicitly models both the adversary's potential influence and the knowledge the adversary gains during real-world execution.

We denote by $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ the output of the environment $\mathcal{Z}$ when interacting with protocol Π and a real-world adversary $\mathcal{A}$. Similarly, $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denotes $\mathcal{Z}$'s output in the ideal world, where it interacts with the ideal functionality $\mathcal{F}$ and an ideal-world adversary represented by a simulator $\mathcal{S}$.

All the entities of our scheme (protocol parties as well as $\mathcal{Z}, \mathcal{A}, \mathcal{S}$) are modeled as instances of PPT Turing machines (ITMs) - the running ITMs, distinguished from each other both by a session identifier (sid) and a party identifier (pid). We consider the synchronous communication model with random oracle access and static corruption of the parties.

The security of our scheme we prove by constructing a PPT simulator $\mathcal{S}$ that can provide an ideal world execution view for environment $\mathcal{Z}$ which is indistinguishable from $\mathcal{Z}$'s view in the real world execution of our protocol for any PPT real/hybrid world adversary $\mathcal{A}$: $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

III. The proposed voting scheme

A. Security properties

In this section, we begin with an informal discussion of the security properties provided by our protocol. We then proceed to the formal UC modeling and the corresponding security proofs.

a) Ballot Privacy (informal): Ballot privacy informally requires that an adversary cannot learn the choices of honest voters or delegates beyond what is revealed by the final tally. In our protocol, this property is primarily ensured by encrypting all ballots under a common public key of the Lifted ElGamal threshold encryption scheme. The public key is jointly generated by the committee using a secure distributed key generation (DKG) protocol. As long as a majority of the committee members are honest, no single party learns the corresponding secret key, and ballot ciphertexts can only be decrypted jointly by the committee after the voting phase has concluded.

Each vote is represented as an encrypted unit vector, where exactly one entry with value "1" corresponds to the selected option or delegate, and all remaining entries are encryptions of zero. The secrecy of this representation follows from the IND-CPA security of the Lifted ElGamal encryption scheme under the DDH assumption and the use of a jointly generated public key. Zero-knowledge arguments of knowledge are used to publicly verify that the ballot is well-formed, namely that it encodes a valid unit vector, without revealing which position is non-zero. Importantly, these arguments also ensure that the voter or delegate actually knows the plaintext values encrypted in the ballot. This prevents ballot copying or replay attacks, in which an adversary could otherwise submit a copied ciphertext without knowledge of its contents, thereby violating ballot privacy.

In addition, each ballot contains a second vector obtained by homomorphically multiplying the original unit vector by the voter's or delegate's stake share and re-encrypting the result. Re-encryption is necessary to prevent disclosure of the stake share via exhaustive search, since stake values lie in a limited range and could otherwise be inferred from the relation between the original and weighted vectors.

A dedicated zero-knowledge argument of knowledge proves that all elements of the original unit vector are multiplied by the same stake value and that the resulting ciphertexts are valid re-encryptions, without revealing either the selected option or the stake share.

The stake shares themselves are hidden using Pedersen commitments under a public commitment key, which is either generated jointly by the participants or fixed so that no party knows the discrete logarithm relation between the generators. Under this setup, the commitments are perfectly hiding and computationally binding under the discrete logarithm assumption. Zero-knowledge range proofs ensure that the committed stake values are non-negative, while a consistency check guarantees that the sum of all committed stake shares equals

the publicly known total stake of the voter or delegate by opening the randomness corresponding to the sum of the commitments. Together, these mechanisms ensure that neither the selected option nor the stake distribution of an honest voter or delegate is revealed to the adversary.

b) Verifiability (informal).: Verifiability informally requires that all ballots cast by honest voters are correctly included in the final tally, and that an adversary cannot influence the election outcome beyond the voting power corresponding to the voters it controls. In particular, no party should be able to introduce malformed ballots, tamper voting power, remove ballots, or manipulate the tally without being detected.

In our protocol, verifiability is achieved by publicly verifying ballot correctness using zero-knowledge arguments of knowledge. These arguments ensure that each ballot encodes a well-formed vote, that stake shares are correctly applied and consistently committed, and that all ballot ciphertexts are encrypted under a common public key, enabling correct and publicly verifiable tallying via threshold decryption by the committee. All ballots are submitted to a public bulletin board, allowing any observer to verify the correctness of the tallying process. As a result, the final tally can be publicly verified to correspond exactly to the aggregation of all valid ballots published on the bulletin board.

Importantly, the use of a bulletin board also provides individual verifiability: each voter or delegate can check that their encrypted ballot has been correctly recorded on the bulletin board and is included in the tally. To exclude re-voting, the bulletin board accepts at most one ballot per participant and election session *sid*; any subsequent ballot associated with the same participant is ignored. Therefore, each voter or delegate can verify that the unique ballot recorded under their identifier is exactly the one that will be processed in the tally.

c) Fairness (informal): Fairness informally requires that no party can obtain information about the election outcome before the voting phase ends. In our protocol, fairness is ensured by encrypting all ballots under a common public key throughout the voting period and by decrypting only the final tally after voting is closed. As long as a majority of the committee members are honest, no party can compute partial results or infer intermediate outcomes, thereby preventing strategic behavior based on early outcome information.

B. Simulation-based security

Our voting scheme involves the following entities: the set of u trustees (voting committee members) $\mathrm{T} = \{T^{(1)}, \ldots, T^{(u)}\}$, the set of m delegates $\mathrm{D} = \{D^{(1)}, \ldots, D^{(m)}\}$ and the set of n voters $\mathrm{V} = \{V^{(1)}, ..., V^{(n)}\}$. We also define $\Pi = \{\Pi^{(1)}, \ldots, \Pi^{(l)}\}$ as the set of l projects (proposals) to vote on.

Table II summarizes the main notation used throughout the paper.

The ideal world execution. In the ideal world, the voting committee T, the voters V, and the delegates D only communicate to an ideal functionality$\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ (here (t, u) is a threshold for trustees) during the execution. The ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ accepts commands from T, V, D. At the same time, it informs the adversary of certain actions that take place and also is influenced by the adversary $\mathcal{S}$ to elicit certain actions. Alg. 1 represents the definition of $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$, which consists of three phases: *Preparation*, *Voting/Delegation*, and *Tally*.

TABLE II
MAIN NOTATION

Symbol	Meaning
$\mathrm{T}, \mathrm{D}, \mathrm{V}, \Pi$	sets of trustees, delegates, voters, and projects
t, u, m, n, l	threshold, number of trustees, delegates, voters, and projects
$\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$	ideal voting functionality
$\mathcal{F}_{\text{BLOCKCHAIN}}$	ideal blockchain functionality
$\mathcal{F}_{\text{DKG}}^{t,u}$	ideal functionality for distributed threshold key generation
$\Pi_{\text{VOTE}}^{t,u,m,n}$	real voting protocol
pk, sk	public encryption key and secret key; sk is used only in simulation
ck	public commitment key
μ	conditional voting power assigned to each delegate before delegated stakes are applied
$s^{(i)}$	square root of the publicly known stake of voter $V^{(i)}$
N	maximal bit-length of a stake-share value
π^{UV}, π^{Exp}, π^{RP}	unit-vector proof, exponentiation correctness proof, and range proof

Preparation phase. During the preparation phase, the trustees $T^{(i)} \in \mathrm{T}$ need to initiate the voting process by sending (INIT, sid) to the ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$. Voting will not start until all trustees have participated in the preparation phase.

Voting/Delegation phase. During the voting/delegation phase, the delegate $D^{(i)} \in \mathrm{D}$ can vote for his choice $v^{(i)}$ by sending $(\text{VOTE}, sid, v^{(i)})$ to the ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$. Note that the voting choice $v^{(i)}$ is leaked only when at least t of the trustees are corrupted. Let $s^{(i)}$ denote the square root of the publicly known stake owned by voter $V^{(i)} \in \mathrm{V}$. For each project, $V^{(i)}$ may either vote directly for his choice $v^{(i)}$ or delegate his voting power to a delegate $D^{(j)} \in \mathrm{D}$. Similarly, when at least t of the trustees are corrupted, $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ leaks the voters' ballots to the adversary $\mathcal{S}$.

To clarify the logic behind the voting procedure, we will outline below the core idea that enables simultaneous voting by both voters and delegates, even though delegates' voting power depends on the outcome of the voters' decisions.

At the beginning of the voting process, each delegate is assumed to hold a fixed *conditional* voting power, denoted by μ (e.g., $\mu = 1000$), which they distribute across proposals. Specifically, a delegate $D^{(i)}$ allocates values $z_j^{(i)}$ to each proposal $\Pi^{(j)}, j \in [l]$, such that: $\sum_{j=1}^{l} z_j^{(i)} = \mu$.

At the final stage, once the actual total delegated stakes $\{d_j^{(i)}\}_{j=1}^{l}$ received by delegate $D^{(i)}$ from voters are known, the values $\{z_j^{(i)}\}_{j=1}^{l}$ are multiplied with them: $\{z_j^{(i)} \cdot d_j^{(i)}\}_{j=1}^{l}$ (in the protocol we exponentiate the encrypted $z_j^{(i)}$ to the power $d_j^{(i)}$, leveraging the homomorphic properties of Lifted ElGamal encryption). The resulting values are the final voting weights of delegate $D^{(i)}$ relatively to the projects $\{\Pi^{(j)}\}_{j=1}^{l}$.

Since delegated stakes are effectively multiplied by μ, the

votes of those voters who did not delegate their stake (i.e., voted YES/NO/ABSTAIN) must also be scaled by the same value μ to ensure consistency (in the protocol, this is implemented by exponentiating the encrypted votes of such voters to the power of μ). This scaling by μ does not affect the voting outcome, since multiplying by a positive constant preserves the ordering induced by the outcome function. Equivalently, the results are computed in units that are μ times smaller.

Tally phase. During the tally phase, the trustees $T^{(i)} \in \mathrm{T}$ send $(\textsc{DelCal}, sid)$ to the ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ to compute and reveal the delegations $\{d_j^{(i)}\}_{j=1}^{l}$ received by each delegate $D^{(i)}$ (the delegations calculation algorithm DelCal is described in Alg. 4). Afterwards, the trustees send $(\textsc{Tally}, sid)$ to $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ to open the final tally. Once at least t trustees have submitted the TALLY request, any party can retrieve the tally result by sending $(\textsc{ReadTally}, sid)$ to $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$. Note that due to the nature of threshold cryptography, the adversary $\mathcal{S}$ may learn the tally result before all honest parties. Hence, the adversary may choose to prevent tally disclosure based on the voting outcome. The tally algorithm TallyAlg is described in Alg. 5.

Algorithm 5 The tally algorithm TallyAlg

Input: a set of the voters $\mathrm{V} = \left\{V^{(1)}, ..., V^{(n)}\right\}$, a set of the delegates $\mathrm{D} = \left\{D^{(1)}, ..., D^{(m)}\right\}$, two sets of ballots, ϕ_1 and ϕ_2, and the delegation δ; conditional voting power μ and number of projects l.
Output: the tally result τ.

Init:
- For each $i \in [l]$, create and initialize $\tau_i^{\texttt{YES}} = 0$, $\tau_i^{\texttt{NO}} = 0$, $\tau_i^{\texttt{ABSTAIN}} = 0$.
- Parse δ as $\left\{\left(D^{(i)};\ \left(d_1^{(i)},\ ...,\ d_l^{(i)}\right)\right)\right\}_{i=1}^{m}$

Tally computation:
- For each ballot $B \in \phi_2$:
  - parse B in the form of $\left(V^{(j)},\ Cast,\ v^{(j)}, s^{(j)}\right)$;
  - parse $v^{(j)}$ in the form of $v^{(j)} = \left(s_i^{(j)},\ \alpha_i^{(j)}\right)_{i=1}^{l}$, where $\alpha_i^{(j)} \in \left\{D^{(1)},\ ...,\ D^{(m)}\right\} \bigcup \{\texttt{"YES"},\ \texttt{"NO"},\ \texttt{"ABSTAIN"}\}$;
  - for each $i \in [l]$, if $\alpha_i^{(j)} \in \{\texttt{"YES"},\ \texttt{"NO"},\ \texttt{"ABSTAIN"}\}$ set $\tau_i^{\alpha_i^{(j)}} := \tau_i^{\alpha_i^{(j)}} + \mu \cdot s_i^{(j)}$.
- For each ballot $B \in \phi_1$:
  - parse B in the form of $\left(D^{(j)},\ Vote,\ v^{(j)}\right)$;
  - parse $v^{(j)}$ in the form of $v^{(j)} = \left(s_i^{(j)},\ \alpha_i^{(j)}\right)_{i=1}^{l}$, where $\alpha_i^{(j)} \in \{\texttt{"YES"},\ \texttt{"NO"},\ \texttt{"ABSTAIN"}\}$;
  - for each $i \in [1,\ l]$, if $\alpha_i^{(j)} \in \{\texttt{"YES"},\ \texttt{"NO"},\ \texttt{"ABSTAIN"}\}$ set $\tau_i^{\alpha_i^{(j)}} := \tau_i^{\alpha_i^{(j)}} + s_i^{(j)} \cdot d_i^{(j)}$.

Output:
- Return $\tau := \{\tau_i\}_{i=1}^{l} = \left\{\left(\tau_i^{\texttt{YES}},\ \tau_i^{\texttt{NO}},\ \tau_i^{\texttt{ABSTAIN}}\right)\right\}_{i=1}^{l}$.

The real/hybrid world execution. In the real/hybrid world, our voting scheme relies on two supporting functionalities: $\mathcal{F}_{\text{BLOCKCHAIN}}$ and $\mathcal{F}_{\text{DKG}}^{t,u}$. We describe them below.

Algorithm 1 The ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$

The ideal functionality $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ interacts with a set of trustees $\mathrm{T} = \left\{T^{(1)}, ..., T^{(u)}\right\}$, a set of voters $\mathrm{V} = \left\{V^{(1)}, ..., V^{(n)}\right\}$, a set of delegates $\mathrm{D} = \left\{D^{(1)}, ..., D^{(m)}\right\}$, a set of projects $\Pi = \left\{\Pi^{(1)}, \ldots, \Pi^{(l)}\right\}$ and the adversary S. It is parameterized by the delegate's and voter's votes validation algorithms validateD (as defined in Alg. 2) and validateV (Alg. 3), by the delegation calculation algorithm DelCal (Alg. 4) and by the tally algorithm TallyAlg (Alg. 5) and variables ϕ_1, ϕ_2, τ, J_1, J_2, J_3, T_1, and T_2. Denote T_{corr} and T_{honest} as the set of corrupted and honest trustees, respectively. Intially, $\phi_1 = \emptyset$, $\phi_2 = \emptyset$, $\delta = \emptyset$, $\tau = \emptyset$, $J_1 = \emptyset$, $J_2 = \emptyset$, $J_3 = \emptyset$.

Preparation:
- Upon receiving $(\textsc{Init}, sid)$ from the trustee $T^{(i)} \in \mathrm{T}$, set $J_1 := J_1 \cup \left\{T^{(i)}\right\}$, and send a notification message $(\textsc{InitNotify}, sid, T^{(i)} \in \mathrm{T})$ to the adversary S.

Voting/Delegation:
- Upon receiving $(\textsc{Vote}, sid, v^{(i)})$ from the delegate $D^{(i)} \in \mathrm{D}$, call validateD $\left(v^{(i)}\right)$. If validateD $\left(v^{(i)}\right) = 0$ or $|J_1| < u$ or there already exists an entry of the form $(D^{(i)}, Vote, \cdot)$ in ϕ_1, ignore the request. Otherwise, record $(D^{(i)}, Vote, v^{(i)})$ in ϕ_1; send a notification message $(\textsc{VoteNotify}, sid, D^{(i)})$ to the adversary S. If $|T_{corr}| \geq t$ then additionally send a message $(\textsc{Leak}, sid, D^{(i)}, Vote, v^{(i)})$ to the adversary S.
(here $v^{(i)}$ contains also a split of some conditional value μ, for example μ=1000)
- Upon receiving $(\textsc{Cast}, sid, v^{(i)}, s^{(i)})$ from the voter $V^{(i)} \in \mathrm{V}$, call validateV $\left(v^{(i)}\right)$. If validateV $\left(v^{(i)}\right) = 0$ or $|J_1| < u$ or there already exists an entry of the form $(V^{(i)}, Cast, \cdot, \cdot)$ in ϕ_2, ignore the request. Otherwise, record $(V^{(i)}, Cast, v^{(i)}, s^{(i)})$ in ϕ_2; send a notification message $(\textsc{CastNotify}, sid, V^{(i)})$ to the adversary S. If $|T_{corr}| \geq t$, then additionally send a message $(\textsc{Leak}, sid, V^{(i)}, Vote, v^{(i)})$ to the adversary S.
(here $v^{(i)}$ contains also a split of $s^{(i)}$)

Tally:
- Upon receiving $(\textsc{DelCal}, sid)$ from the trustee $T^{(i)} \in \mathrm{T}$, set $J_2 := J_2 \cup \left\{T^{(i)}\right\}$, and send a notification message $(\textsc{DelCalNotify}, sid, T^{(i)})$ to the adversary S.
- If $|J_2 \cap T_{honest}| + |T_{corr}| \geq t$, send $(\textsc{LeakDel}, sid, \text{DelCal}\,(\mathrm{D}, \phi_2))$ to the adversary S.
- If $|J_2| \geq t$, set $\delta \leftarrow \text{DelCal}\,(\mathrm{D}, \phi_2)$.
- Upon receiving $(\textsc{Tally}, sid)$ from the trustee $T^{(i)} \in \mathrm{T}$, set $J_3 := J_3 \cup \left\{T^{(i)}\right\}$, and send a notification message $(\textsc{TallyNotify}, sid, T^{(i)})$ to the adversary S.
- If $|J_3 \cap T_{honest}| + |T_{corr}| \geq t$, send $(\textsc{LeakTally}, sid, \text{TallyAlg}\,(\mathrm{V},\ \mathrm{D},\ \phi_1,\ \phi_2,\ \delta))$ to the adversary S.
- If $|J_3| \geq t$, set $\tau \leftarrow \text{TallyAlg}\,(\mathrm{V},\ \mathrm{D},\ \phi_1,\ \phi_2,\ \delta)$.
- Upon receiving $(\textsc{ReadTally}, sid)$ from any party, if $\delta = \emptyset \wedge \tau = \emptyset$ ignore the request. Otherwise, return $(\textsc{ReadTallyReturn}, sid, (\delta, \tau))$ to the requester.

Algorithm 2 The delegate's vote validation algorithm validateD

Input: Delegate's vote $v^{(j)} = \left(s_i^{(j)},\ \alpha_i^{(j)}\right)_{i=1}^{l}$; conditional voting power μ and number of projects l.
Output: 0 or 1 for invalid or valid.

Algorithm: If $\sum_{i=1}^{l} s_i^{(j)} = \mu \wedge$ for each $i \in [l]$: $\{0 \leq s_i^{(j)} \leq \mu \wedge \alpha_i^{(j)} \in \{$"YES", "NO", "ABSTAIN"$\}\}$ (the last one is checked as Unit Vector proof), label=

- then validateD $\left(v^{(i)}\right) = 1$,
- else validateD $\left(v^{(i)}\right) = 0$.

Output:

- Return validateD $\left(v^{(i)}\right)$.

Algorithm 3 The voter's vote validation algorithm validateV

Input: Voter's vote $v^{(j)} = \left(s_i^{(j)},\ \alpha_i^{(j)}\right)_{i=1}^{l}$; stake value $s^{(j)}$ and number of projects l.
Output: 0 or 1 for invalid or valid.

Algorithm: If $\sum_{i=1}^{l} s_i^{(j)} = s^{(j)} \wedge$ for each $i \in [l]$: $\{0 \leq s_i^{(j)} \leq s^{(j)} \wedge \alpha_i^{(j)} \in \left\{D^{(1)}, ..., D^{(m)}\right\} \bigcup \{$"YES", "NO", "ABSTAIN"$\}\}$ (the last one is checked as Unit Vector proof), label=

- then validateV $\left(v^{(i)}\right) = 1$,
- else validateV $\left(v^{(i)}\right) = 0$.

Output:

- Return validateV $\left(v^{(i)}\right)$.

Algorithm 4 Algorithm DelCal

Input: a set of the delegates $\mathrm{D} = \left\{D^{(1)}, ..., D^{(m)}\right\}$, and a set of ballots ϕ_2; number of projects l
Output: the delegation result δ.

Init:

- For $i \in [1,\ m]$, $s \in [1,\ l]$ create and initiate $d_s^{(i)} = 0$.

Delegation interpretation:

- For each ballot $B \in \phi_2$:
  - parse B in the form of $\left(V^{(j)}, Cast, v^{(j)}\right)$;
  - parse $v^{(j)}$ in the form of $v^{(j)} = \left(s_i^{(j)},\ \alpha_i^{(j)}\right)_{i=1}^{l}$, where $\alpha_i^{(j)} \in \left\{D^{(1)}, ..., D^{(m)}\right\} \bigcup \{$"YES", "NO", "ABSTAIN"$\}$;
  - for each $i \in [1,\ l]$, if $\alpha_i^{(j)} \in \left\{D^{(1)}, ..., D^{(m)}\right\}$ then $d_i^{\alpha_i^{(j)}} := d_i^{\alpha_i^{(j)}} + s_i^{(j)}$.

Output:

- Return $\delta := \left\{\left(D^{(i)};\ \left(d_1^{(i)}, ..., d_l^{(i)}\right)\right)\right\}_{i=1}^{m}$.

The blockchain ideal functionality. We use the functionality $\mathcal{F}_{\text{BLOCKCHAIN}}$ [33] depicted in Alg. 6 to define an append-only ledger commonly implemented by many blockchain protocols. $\mathcal{F}_{\text{BLOCKCHAIN}}$ interacts with a set of parties $\mathcal{P} := \{P_1, \ldots, P_u\}$ and models transactions validation with a successor relationship parameter – succ. The stored data is identified using the id parameter, and in addition to the READ/WRITE interfaces, $\mathcal{F}_{\text{BLOCKCHAIN}}$ provides an interface labeled "$\in$" that allows parties to check whether a given data item is contained in the blockchain.

Algorithm 6 Ideal functionality $\mathcal{F}_{\text{BLOCKCHAIN}}[\text{succ}]$

The parameter succ defines the validity of new items. A new item can only be appended to the storage if the evaluation of succ outputs 1.
Parameter: successor relationship succ : $\{0,1\}^* \times \{0,1\}^* \to \{0,1\}$
On receive (INIT, sid): Storage := $\emptyset$
On receive (READ, id, sid): output Storage[id], or $\perp$ if not found
On receive (WRITE, id, inp, sid) from $\mathcal{P}$: left=1em

- let val := Storage[id], set to $\perp$ if not found
- **if** succ(val, inp) $= 1$ **then**
  - Storage[id] := val$\|(inp, \mathcal{P})$
  - output (RECEIPT, id)
- **else** output (REJECT, id)

On receive ("$\in$", id, val): left=1em

- **if** $val \in$ Storage[id] **then** output true **else** output false

The distributed key generation ideal functionality. We use the functionality $\mathcal{F}_{\text{DKG}}^{t,u}$ [34] depicted in Alg. 7 for distributed threshold key generation of the public key pk, which is used in our voting scheme to encrypt voters' and delegates' ballots with a Lifted ElGamal encryption scheme. $\mathcal{F}_{\text{DKG}}^{t,u}$ interacts with a set of trustees $T := \{T_1, \ldots, T_u\}$ to generate a public key pk and deal the corresponding secret key sk among them with a (t, u)-threshold.

Algorithm 7 Ideal functionality $\mathcal{F}_{\text{DKG}}^{t,u}$

The functionality $\mathcal{F}_{\text{DKG}}^{t,u}$ interacts with a set of trustees $T := \{T_1, \ldots, T_u\}$, set of parties $\mathcal{P}$, and the adversary $\mathcal{A}$. It is parameterised with variable $\mathcal{J}$ and algorithms KeyGen() and Deal(). Initially, set $\mathcal{J} := \emptyset$.

- Upon receiving (KEYGEN, sid) from $T_j \in T$, set $\mathcal{J} := \mathcal{J} \cup \{T_j\}$.
- If $|\mathcal{J}| = u$, generate a public key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and compute $(\mathsf{sk}_1, \ldots, \mathsf{sk}_u) \leftarrow \mathsf{Deal}(t, u, \mathsf{sk})$. For $j \in [u]$, send message (PRIVKEY, sid, sk_j) to T_j.
- Upon receiving (READPK, sid) from any party $p \in \mathcal{P}$, if pk is not defined yet, ignore the request. Otherwise, it sends (PUBLICKEY, sid, pk) to the requestor.

Definition III.1 (UC-realization)**.** *Let* $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ *denote the output of the environment* $\mathcal{Z}$ *when it interacts with the parties executing protocol* Π *and a real-world adversary* $\mathcal{A}$*. Let* $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ *denote the output of* $\mathcal{Z}$ *when it interacts with the ideal functionality* $\mathcal{F}$*, the ideal-world adversary (simulator)* $\mathcal{S}$*, and dummy parties. We say that protocol* Π UC-realizes $\mathcal{F}$ *if for every PPT adversary* $\mathcal{A}$*, there exists a PPT simulator* $\mathcal{S}$ *such that for every PPT environment* $\mathcal{Z}$*, it holds that:* $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

C. The voting scheme

In what follows, we will use the encoding of votes (ballots), which allows for a detailed description of votes encryption and proofs creation. As the vote format differs for voters and delegates, we will use two different algorithms, encodeV and encodeD, defined in Alg. 8 and Alg. 9, respectively.

In our scheme, we encode the values of $\alpha_i^{(j)}$ into unit vectors. For delegates' ballots, we encode $\alpha_i^{(j)}$ as a 3-bit unit vector: "YES" as $(1,0,0)$, "NO" as $(0,1,0)$ and "ABSTAIN" as $(0,0,1)$.

For Voters' ballots, we encode $\alpha_i^{(j)}$ as $m+3$ bits unit vector: "YES" as 1 on the $m+1$ place (all other elements are zero); "NO" as 1 on the $m+2$ place; "ABSTAIN" as 1 on the $m+3$ place; vector with 1 on the i-th place, $i \in [m]$ (all other elements are zero) means delegation to the delegate $D^{(i)}$.

Each voter's or delegate's ballot is posted to $\mathcal{F}_{\text{BLOCKCHAIN}}$ under a unique identifier derived from $(role, i)$. Only the first valid ballot posted under this identifier is considered by the voting protocol, while any subsequent ballot with the same identifier is ignored. This ensures that re-voting is not allowed and that each voter or delegate can individually verify that their unique recorded ballot is the one included in the tally.

Stake values $s_j^{(i)}$ are considered to be represented with the elements of some residue ring Z_p for the corresponding prime p.

The voting protocol $\Pi_{Vote}^{t,u,m,n}$ is described in Alg. 12, while the supporting algorithm for creating encrypted ballots of voters and delegates is presented in Alg. 10, and the corresponding verification algorithm is shown in Alg. 11.

Algorithm 8 The encoding algorithm encodeV

Input: Voter's choice $\alpha \in \{D^{(1)}, ..., D^{(m)}\} \cup \{\texttt{"YES"}, \texttt{"NO"}, \texttt{"ABSTAIN"}\}$.
Output: unit vector $\mathbf{e}^{(m+3)}$.

Algorithm: Set $\mathbf{e}^{(m+3)}$ to:
- $(\delta_{1k}, ..., \delta_{mk}, 0, 0, 0)$ if $\alpha = D^{(k)}$, where $k \in [m]$ and δ_{ik} for $i \in [m]$ is the Kronecker delta function;
- $(\underbrace{0, ..., 0}_{m}, 1, 0, 0)$ if $\alpha =$ "YES";
- $(\underbrace{0, ..., 0}_{m}, 0, 1, 0)$ if $\alpha =$ "NO";
- $(\underbrace{0, ..., 0}_{m}, 0, 0, 1)$ if $\alpha =$ "ABSTAIN".

Output:
- Return $\mathbf{e}^{(m+3)}$.

Algorithm 9 The encoding algorithm encodeD

Input: Delegate's choice $\alpha \in \{\texttt{"YES"}, \texttt{"NO"}, \texttt{"ABSTAIN"}\}$.
Output: unit vector $\mathbf{e}^{(3)}$.

Algorithm: Set $\mathbf{e}^{(3)}$ to:
- $(1, 0, 0)$ if $\alpha =$ "YES";
- $(0, 1, 0)$ if $\alpha =$ "NO";
- $(0, 0, 1)$ if $\alpha =$ "ABSTAIN".

Output:
- Return $\mathbf{e}^{(3)}$.

Algorithm 10 Ballot encryption algorithm $\mathsf{EncBallot}_{BT}$

Input: ballot type $BT \in \{\texttt{"V"}, \texttt{"D"}\}$; a set of votes for selected projects $v = \{(s_j, \alpha_j)\}_{j=1}^{l}$; maximal size in bits of a stake share value N; encryption public key pk; commitment key ck.
Output: encrypted ballot B.

Algorithm:
- Initialize the set of encrypted votes $EV := \emptyset$ and the randomness sum $t_{\mathsf{sum}} := 0$.
- Set $(\mathsf{encode}, \theta)$ to
  - $(\mathsf{encodeV},\ m+3)$ if $BT =$ "V";
  - $(\mathsf{encodeD},\ 3)$ otherwise.
- Set $\mathbf{0}^{(\theta)}$ as a θ-sized vector of all zeroes.
- For each $j \in [l]$ do the following steps:
  - Compute the unit vector $\mathbf{e}_j^{(\theta)} := \mathsf{encode}(\alpha_j)$.
  - Sample randomness $\mathbf{r}_j^{(\theta)} \leftarrow \mathbb{Z}_q^{\theta}$.
  - Compute Lifted ElGamal encryption of the unit vector: $\mathbf{w}_j^{(\theta)} := \mathsf{Enc}_{\mathsf{pk}}(\mathbf{e}_j^{(\theta)}, \mathbf{r}_j^{(\theta)})$.
  - Generate NIZK proof of unit vector correctness: $\pi_j^{\mathsf{UV}} := \mathsf{NIZK}_{UV}.\mathsf{Prove}((g, \mathsf{ck}, \mathsf{pk}, \theta, \mathbf{w}_j^{(\theta)}), (\mathbf{e}_j^{(\theta)}, \mathbf{r}_j^{(\theta)}))$.
  - Sample $t_j \leftarrow \mathbb{Z}_q$ and update $t_{\mathsf{sum}} := t_{\mathsf{sum}} + t_j$.
  - Compute Pedersen commitment: $C_j := \mathsf{Com}_{\mathsf{ck}}(s_j, t_j)$.
  - Generate NIZK range proof: $\pi_j^{\mathsf{RP}} := \mathsf{NIZK}_{RP}.\mathsf{Prove}((g, \mathsf{ck}, N, C_j), (s_j, t_j))$.
  - Sample randomness $\mathbf{p}_j^{(\theta)} \leftarrow \mathbb{Z}_q^{\theta}$.
  - Compute exponentiated and re-encrypted vector of ciphertexts: $\mathbf{W}_j^{(\theta)} := \mathsf{Exp}(\mathbf{w}_j^{(\theta)}, s_j \mathbf{1}^{(\theta)}) \cdot \mathsf{Enc}_{\mathsf{pk}}(\mathbf{0}^{(\theta)}, \mathbf{p}_j^{(\theta)})$.
  - Generate NIZK proof of correct exponentiation and re-encryption: $\pi_j^{\mathsf{Exp}} := \mathsf{NIZK}_{Exp}.\mathsf{Prove}((g, \mathsf{ck}, \mathsf{pk}, \theta, \mathbf{w}_j^{(\theta)}, \mathbf{W}_j^{(\theta)}, C_j), (s_j, t_j, \mathbf{p}_j^{(\theta)}))$.
  - Append to encrypted votes: $EV := EV \cup \left\{\left(\mathbf{w}_j^{(\theta)}, \mathbf{W}_j^{(\theta)}, C_j, \pi_j^{\mathsf{UV}}, \pi_j^{\mathsf{Exp}}, \pi_j^{\mathsf{RP}}\right)\right\}$.

Return: $B := (EV,\ t_{\mathsf{sum}})$.

Algorithm 11 Ballot verification algorithm $\mathsf{VerifyBallot}_{BT}$

Input: ballot type $BT \in \{\texttt{"V"}, \texttt{"D"}\}$; encrypted ballot B, voter's/delegate's total stake s, maximal size in bits of a stake share value N; encryption public key pk; commitment key ck.
Output: verification status $\in \{0, 1\}$.

Algorithm:
- Set $\theta := m+3$ if $BT =$ "V", otherwise $\theta := 3$; initialize $C_{\text{prod}} := 1$.
- Parse encrypted ballot B as $\left(\left\{\left(\mathbf{w}_j^{(\theta)}, \mathbf{W}_j^{(\theta)}, C_j, \pi_j^{\mathsf{UV}}, \pi_j^{\mathsf{Exp}}, \pi_j^{\mathsf{RP}}\right)\right\}_{j=1}^{l},\ t_{\mathsf{sum}}\right)$.
- For each $j \in [l]$ do the following steps:
  - Verify all associated NIZK proofs:
    - $\mathsf{NIZK}_{UV}.\mathsf{Verify}((g, \mathsf{ck}, \mathsf{pk}, \theta, \mathbf{w}_j^{(\theta)}), \pi_j^{\mathsf{UV}}) = 1$;
    - $\mathsf{NIZK}_{Exp}.\mathsf{Verify}((g, \mathsf{ck}, \mathsf{pk}, \theta, \mathbf{w}_j^{(\theta)}, \mathbf{W}_j^{(m+3)}, C_j), \pi_j^{\mathsf{Exp}}) = 1$;
    - $\mathsf{NIZK}_{RP}.\mathsf{Verify}((g, \mathsf{ck}, N, C_j), \pi_j^{\mathsf{RP}}) = 1$.

    If any of the above verifications fails, skip all further steps and **return 0**.
  - Update $C_{\text{prod}} := C_{\text{prod}} \cdot C_j$.
- Verify the condition $C_{\text{prod}} = g^s \cdot \mathsf{ck}^{t_{\mathsf{sum}}}$ and if it fails **return 0**, otherwise **return 1**.

D. Security

The security of our voting protocol is analyzed in the UC framework. The corresponding security guarantee is formalized in Theorem 1.

Theorem 1. *Let $t, u, m, n = \text{poly}(\lambda)$ and $u/2 < t \leq u$. Protocol $\Pi_{\text{VOTE}}^{t,u,m,n}$ described in Alg. 12 UC-realizes $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ in the $\{\mathcal{F}_{\text{BLOCKCHAIN}}, \mathcal{F}_{\text{DKG}}^{t,u}\}$-hybrid world against static corruption under the DDH assumption in the random oracle model.*

Algorithm 12 The voting protocol $\Pi_{Vote}^{t,u,m,n}$

CRS: commitment key ck.

Preparation phase:

– Upon receiving (INIT, sid) from the environment Z, each trustee $T^{(j)}$, $j \in [u]$, sends (KEYGEN, sid) to $\mathcal{F}_{\text{DKG}}^{t,u}$ to generate election key pk and obtain the corresponding partial private key sk_j from $\mathcal{F}_{\text{DKG}}^{t,u}$.

Voting/Delegation phase:

– Upon receiving $(\text{VOTE}, sid, v^{(i)})$ from the environment Z, each delegate $D^{(i)}$, $i \in [m]$, does the next:
  – Send (READPK, sid) to $\mathcal{F}_{\text{DKG}}^{t,u}$, obtaining $(\text{PUBLICKEY}, sid, \mathsf{pk})$.
  – Compute encrypted ballot $B := \mathsf{EncBallot}_{\text{D}}(v^{(i)}, N, \mathsf{pk}, \mathsf{ck})$ and post B to $\mathcal{F}_{\text{BLOCKCHAIN}}$.

– Upon receiving $(\text{VOTE}, sid, v^{(i)})$ from the environment Z, each voter $V^{(i)}$, $i \in [n]$, does the next:
  – Send (READPK, sid) to $\mathcal{F}_{\text{DKG}}^{t,u}$, obtaining $(\text{PUBLICKEY}, sid, \mathsf{pk})$.
  – Compute encrypted ballot $B := \mathsf{EncBallot}_{\text{V}}(v^{(i)}, N, \mathsf{pk}, \mathsf{ck})$ and post B to $\mathcal{F}_{\text{BLOCKCHAIN}}$.

Tally phase:

– Upon receiving (DELCAL, sid) from the environment $\mathcal{Z}$, each trustee $T^{(k)}$, $k \in [u]$, does the next:
  – Initialize $AV := \emptyset$ and $\{\mathbf{A}_j^{(m+3)}\}_{j=1}^{l} := \{\mathbf{1}_j^{(m+3)}\}_{j=1}^{l}$.
  – Fetch the voters' ballots from $\mathcal{F}_{\text{BLOCKCHAIN}}$ into B_{V}.
  – For each ballot $B_{\text{V}}^{(r)} \in B_{\text{V}}$ in the order in which it was posted to $\mathcal{F}_{\text{BLOCKCHAIN}}$:
    * Extract from $B_{\text{V}}^{(r)}$ the identity of the corresponding voter $V^{(i)}$.
    * If $V^{(i)} \in AV$ or $\mathsf{VerifyBallot}_{\text{V}}(B_{\text{V}}^{(r)}, s^{(i)}, N, \mathsf{pk}, \mathsf{ck}) = 0$, discard the ballot $B_{\text{V}}^{(r)}$ and skip to the next ballot.
    * Update $AV := AV \cup \{V^{(i)}\}$.
    * Extract $\{\mathbf{W}_j^{(m+3)}\}_{j=1}^{l}$ from $B_{\text{V}}^{(r)}$.
    * For each $j \in [l]$: update $\mathbf{A}_j^{(m+3)} := \mathbf{A}_j^{(m+3)} \circ \mathbf{W}_j^{(m+3)}$.
  – For each $j \in [l]$: parse $\mathbf{A}_j^{(m+3)}$ as $(\mathbf{D}_j^{(m)}, \mathbf{V}_j^{(3)})$ and jointly decrypt $\mathbf{D}_j^{(m)}$ to $\mathbf{d}_j^{(m)}$.
  – Post the delegations $\{\mathbf{d}_j^{(m)}\}_{j=1}^{l}$ on $\mathcal{F}_{\text{BLOCKCHAIN}}$.

– Upon receiving (TALLY, sid) from the environment $\mathcal{Z}$, each trustee $T^{(k)}$, $k \in [u]$, does the next ($\mathbf{V}_j^{(3)}$ *and* $\mathbf{d}_j^{(m)}$ *are already initialized during the* DELCAL *processing phase*):
  – Initialize $AD := \emptyset$.
  – For each project, scale the direct votes of voters by μ: $\mathbf{V}_j^{(3)} := \mathsf{Exp}(\mathbf{V}_j^{(3)}, \mu\mathbf{1}^{(3)})$, $j \in [l]$.
  – Fetch the delegates' ballots from $\mathcal{F}_{\text{BLOCKCHAIN}}$ into B_{D}.
  – For each ballot $B_{\text{D}}^{(r)} \in B_{\text{D}}$ in the order in which it was posted to $\mathcal{F}_{\text{BLOCKCHAIN}}$:
    * Extract from $B_{\text{D}}^{(r)}$ the identity of the corresponding delegate $D^{(i)}$.
    * If $D^{(i)} \in AD$ or $\mathsf{VerifyBallot}_{\text{D}}(B_{\text{D}}^{(r)}, \mu, N, \mathsf{pk}, \mathsf{ck}) = 0$, discard the ballot $B_{\text{D}}^{(r)}$ and skip to the next ballot.
    * Update $AD := AD \cup \{D^{(i)}\}$.
    * Extract $\{\mathbf{W}_j^{(3)}\}_{j=1}^{l}$ from $B_{\text{D}}^{(r)}$.
    * For each $j \in [l]$:
      · Set the delegated to $D^{(i)}$ in project Π_j stake ω as the i-th element of $\mathbf{d}_j^{(m)}$.
      · Update $\mathbf{V}_j^{(3)} := \mathbf{V}_j^{(3)} \circ \mathsf{Exp}(\mathbf{W}_j^{(3)}, \omega\mathbf{1}^{(3)})$.
  – For each $j \in [l]$: jointly decrypt $\mathbf{V}_j^{(3)}$ to $\mathbf{v}_j^{(3)}$.
  – Post the voting results $\{\mathbf{v}_j^{(3)}\}_{j=1}^{l}$ on $\mathcal{F}_{\text{BLOCKCHAIN}}$.

– Upon receiving $(\text{READTALLY}, sid)$ from the environment $\mathcal{Z}$, any party P does the next:
  – Fetch from $\mathcal{F}_{\text{BLOCKCHAIN}}$ the delegations into $\delta = \{\mathbf{d}_j^{(m)}\}_{j=1}^{l}$ and voting results into $\tau = \{\mathbf{v}_j^{(3)}\}_{j=1}^{l}$.
  – Send $(\text{READTALLYRETURN}, sid, (\delta, \tau))$ to the environment $\mathcal{Z}$.

The proof of Theorem 1 is given in Appendix D.

IV. PERFORMANCE

We implemented the prototype [2] of the proposed voting scheme in Rust. The implementation is based on the *catalyst-core* [35] codebase and includes primitives and algorithms needed for our voting protocol, together with unit tests, benchmarks, and simulation of the voting process. The underlying cryptosystem is the *Ristretto255* group on the Elliptic Curve *Curve25519*. Benchmarking was performed on a workstation with an Intel Core i7-1165G7 @2.80GHz and 32GB of RAM, running Ubuntu 20.04.4 LTS.

In Table III we present benchmarks for the voters' and delegates' ballot creation/validation time and their ballot size, as well as overall traffic, all depending on the number of projects. The provided results are for our voting protocol run with 64 voters and 64 delegates, where each voter and delegate votes for the number of projects specified in the first column of the table.

TABLE III
BENCHMARKS FOR THE PROTOCOL RUN WITH 64 DELEGATES

Proj.	Voter's Ballot			Delegate's Ballot			Total Traffic (MB)
	Create (ms)	Valid. (ms)	Size (KB)	Create (ms)	Valid. (ms)	Size (KB)	
1	18	13	11	4.43	1.6	1.8	0.8
2	36	25	22	9	3	3.5	1.6
4	73	50	44	18	6	7	3.2
8	144	102	88	37	12	14	6.4
16	291	205	176	72	25	28	12.7
32	587	410	351	146	48	55	25.4
64	1245	843	702	304	99	110	50.8
128	2357	1645	1402	582	194	220	101.5

As shown in the table, the delegate's ballot creation/validation time, as well as its size, are always smaller than those of the voter's ballot. This is because delegates are only allowed to select one of the fixed Yes/No/Abstain options, whereas voters may either vote directly or delegate their voting power on each project to one of 64 delegates (as in the voting configuration we used for benchmarking). Consequently, each voter's ballot contains encrypted unit vectors of length 64+3 per project, while these vectors are always of length 3 for the delegates, resulting in a smaller size and faster processing of the delegates' ballots.

Therefore, it's important to note that, unlike the delegate's ballot, the voter's ballot size and processing time depend not only on the number of projects but also on the number of delegates, linearly. This dependency is evident in Table IV, which shows benchmark results for voting with 64 voters and 128 delegates. Compared to the results in Table III, the voter's ballot creation and validation time, as well as its size, increased by approximately a factor of 1.8, whereas the delegate's ballot did not change.

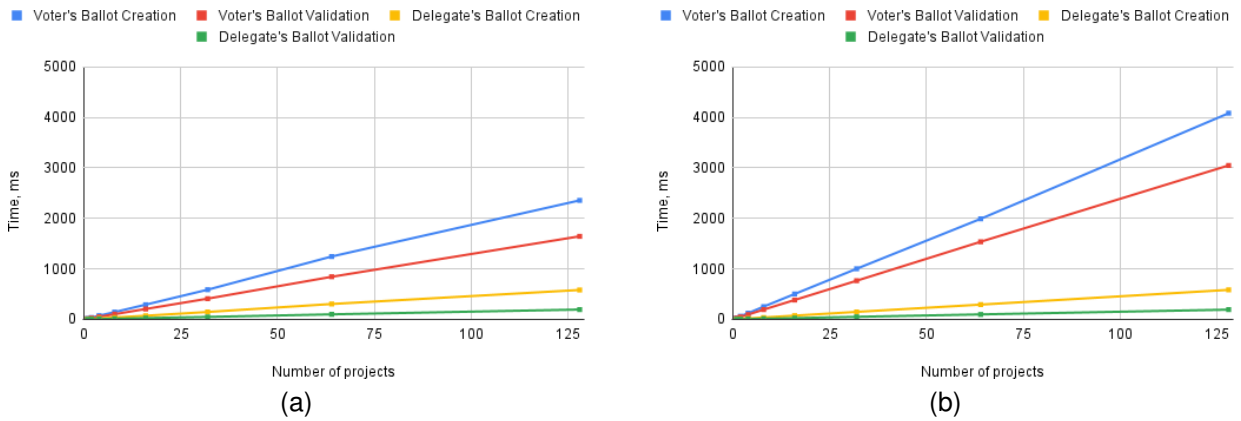

Fig. 1. Ballot processing time (a) 64 delegates (b) 128 delegates

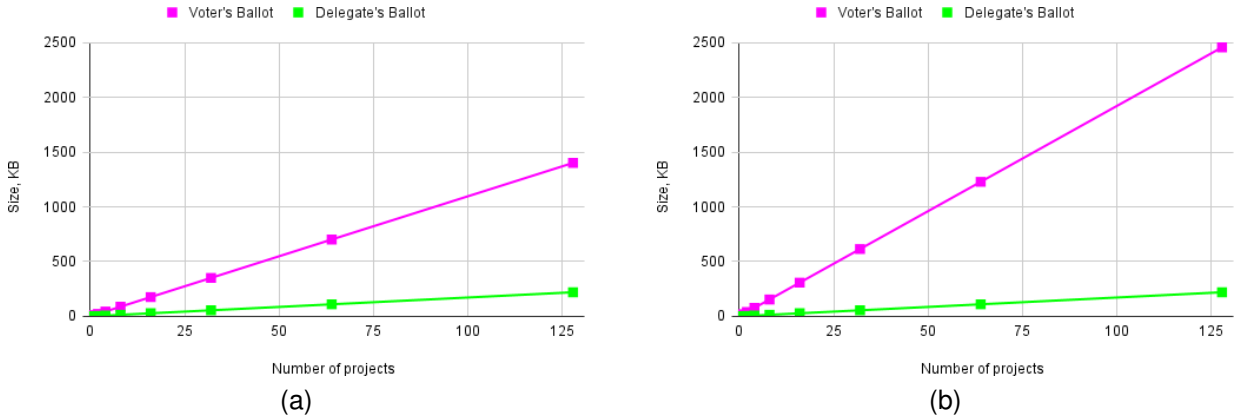

Fig. 2. Ballot size for (a) 64 delegates and (b) 128 delegates

TABLE IV
BENCHMARKS FOR THE PROTOCOL RUN WITH 128 DELEGATES

Proj.	Voter's Ballot			Delegate's Ballot			Total Traffic (MB)
	Create (ms)	Valid. (ms)	Size (KB)	Create (ms)	Valid. (ms)	Size (KB)	
1	31	23	19	4.5	1.5	1.8	1.4
2	62	47	39	9.6	3	3.5	2.9
4	122	94	77	18	6	7	5.7
8	252	193	154	37	12	14	11.3
16	502	383	308	74	24	28	22.7
32	1002	765	615	148	50	55	45.3
64	1991	1537	1229	292	98	110	90.6
128	4084	3050	2458	584	192	220	181.2

We illustrate with Figures 1(a) and 1(b) how the creation/validation time of the voter's and delegate's ballot scales with the number of projects. As shown, both parameters exhibit linear growth. Additionally, the processing of delegates' ballots is consistently faster due to the reasons discussed above.

Figs. 2(a) and 2(b) show that both voter and delegate ballot sizes grow linearly with the number of projects. The figures also indicate that the delegate ballot size is independent of the number of delegates.

Finally, Figs. 3(a) and 3(b) show that the overall traffic generated by our voting scheme grows linearly with the number of projects. Since voter ballots account for most of this traffic, the overall traffic also grows linearly with the number of delegates.

To compare the performance of our scheme with the voting scheme from [5], which is currently deployed in the Cardano Catalyst governance system, we additionally evaluate the overall communication volume using the same parameters as in the performance evaluation of [5]: 5000 voters and 50 delegates in a voting setting with two projects. The resulting overall traffic is 88 MB, corresponding to 44 MB per project, whereas the reported traffic in [5] is 20 MB per project. Since our voting scheme employs two unit vectors per project, the approximately twofold increase in traffic volume is expected. Given that voting periods in modern treasury systems typically span several weeks, the resulting blockchain space overhead appears acceptable for practical deployment.

In practice, the cost of deploying such a scheme in a DAO setting depends not only on the number of voters, delegates, and projects, but also on the size of posted ciphertexts and zero-knowledge proofs, as well as on ledger-specific factors such as transaction size limits, storage costs, and on-chain execution fees. In our construction, the dominant contributors to this cost are the encrypted unit vectors and the accompanying proofs of ballot well-formedness and stake consistency. Further reductions in proof size and verification cost may be

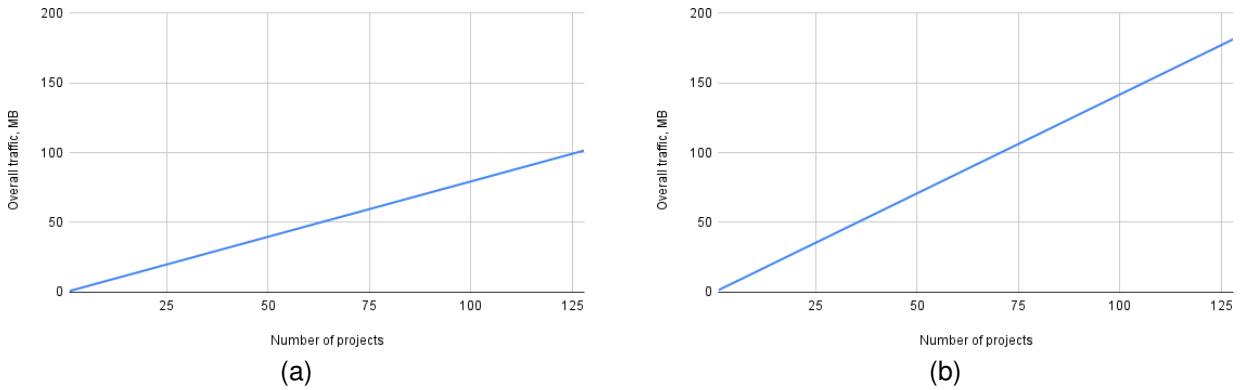

Fig. 3. Overall traffic for (a) 64 delegates and (b) 128 delegates

achievable through aggregated range-proof techniques or alternative compact range-proof systems. Since our construction is blockchain-agnostic and modeled via ideal functionalities, we do not instantiate these costs for any particular ledger. Such factors mainly influence concrete implementation choices while leaving the main protocol design and its UC security guarantees unchanged.

V. Conclusion

In this work, we propose a scalable collaborative decision-making scheme for blockchain systems that supports liquid democracy under the quadratic voting model, while ensuring verifiability, privacy, and fairness of the voting process. The scheme is scalable in the sense that the ballot size, ballot creation and verification time, and overall traffic volume all grow linearly with the number of projects and delegation options. We support our design with both benchmark results and formal security proofs, demonstrating its applicability to modern blockchain-based governance systems.

Future research includes optimizing ballot size and communication overhead through more compact zero-knowledge proof constructions tailored to the quadratic voting setting. In particular, reducing ballot representation to a more succinct form is an interesting direction for further improvement.

At a more general level, while this work adopts a blockchain-agnostic formulation, future research may explore adaptations of the proposed protocol to specific ledger execution models, as well as extensions to other flexible split-voting power schemes beyond quadratic voting.

References

[1] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," Cryptology ePrint Archive, Report 2000/067, 2000, https://eprint.iacr.org/2000/067.

[2] "Split stake voting prototype," https://github.com/input-output-hk/catalyst-split-stake-voting-prototype.

[3] C. Ovezik, D. Karakostas, and A. Kiayias, "Sok: A stratified approach to blockchain decentralization," 2024, https://arxiv.org/pdf/2211.01291.

[4] D. W. E. Allen, C. Berg, and A. M. Lane, "The blockchain treasury governance dilemma," 2022, https://onlinelibrary.wiley.com/doi/pdf/10.1111/rego.12659.

[5] B. Zhang, R. Oliynykov, and H. Balogun, "A treasury system for cryptocurrencies: Enabling better collaborative intelligence," in *The Network and Distributed System Security Symposium (NDSS '19)*, 2019, https://eprint.iacr.org/2018/435.pdf.

[6] L. Kovalchuk, M. Rodinko, R. Oliynykov, A. Nastenko, D. Kaidalov, and K. Nelson, "Enhancing decentralization in proof-of-stake blockchains through quadratic voting," 2025.

[7] E. A. Posner and E. G. Weyl, *Quadratic Voting as Efficient Corporate Governance*, 2014, pp. 251–272.

[8] K. P. Nelson, J. Attieh, and I. Oliveira, "Simulating the quality of community decisions (no. snet df4b-rfp2)," 2024, https://docs.google.com/document/d/1XfJkJvydFCE7-rwsDSaHTa2MaR86FoUhyjnb1X1G0ZA/edit?usp=sharing.

[9] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *CRYPTO '92*, vol. 740, 1993, pp. 89–105, https://chaum.com/publications/Wallet_Databases.pdf.

[10] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO '91*. Springer Berlin Heidelberg, 1991, pp. 129–140, https://link.springer.com/content/pdf/10.1007%2F3-540-46766-1_9.pdf.

[11] M. Nejadgholi, N. Yang, and J. Clark, "Short paper: Ballot secrecy for liquid democracy," 2021, http://dx.doi.org/10.1007/978-3-662-63958-0_26.

[12] Z. Yin, B. Zhang, A. Nastenko, R. Oliynykov, and K. Ren, "A scalable coercion-resistant voting scheme for blockchain decision-making," pp. 1–17, 2026, https://doi.org/10.1109/TDSC.2026.3651473.

[13] D. Karoukis, "Publicly verifiable delegative democracy with secret voting power," 2023, https://arxiv.org/pdf/2302.14421.

[14] A. Kiayias, V. Teague, and O. S. T. Litos, "Privacy preserving opinion aggregation," 2022, https://eprint.iacr.org/2022/760.pdf.

[15] S. Venugopalan, I. Stancikova, and I. Homoliak, "Always on voting: A framework for repetitive voting on the blockchain," 2023, https://arxiv.org/pdf/2107.10571.

[16] B. Zhang and H.-S. Zhou, "Statement voting," 2017, https://eprint.iacr.org/2017/616.pdf.

[17] R. Cramer, R. Gennaro, and B. Schoenmakers, "Optimally efficient multi-authority election scheme," in *In Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1997, pp. 103–118, https://doi.org/10.1007/3-540-69053-0_9.

[18] G. Dini, "A secure and available electronic voting service for a large-scale distributed system," in *Future Gener. Comput. Syst. 19, 1*, 2003, pp. 69–85, https://doi.org/10.1016/S0167-739X(02)00109-7.

[19] A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections," in *AUSCRYPT'92, Workshop on the Theory and Application of Cryptographic Techniques, Gold Coast, Queensland, Australia, December 13-16*, 1992, pp. 244–251, https://doi.org/10.1007/3-540-57220-1_66.

[20] S. Park and R. L. Rivest, "Towards secure quadratic voting," in *Public choice*, 2017, pp. 151–175, https://people.csail.mit.edu/rivest/pubs/PR17.pdf.

[21] S. Lalley and E. G. Weyl, "Quadratic voting: How mechanism design can radicalize democracy," 2018, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2003531.

[22] Z. Zhou, Z. Zhang, F. Hao, B. Zheng, and Z. Masyhur, "Qv-net: Decentralized self-tallying quadratic voting with maximal ballot secrecy," 2025, https://doi.org/10.1145/3719027.3744810.

[23] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," in *Journal of the ACM (JACM), Volume 39, Issue 4*, 1992, pp. 859–868, https://doi.org/10.1145/146585.146605.

[24] O. Pereira and P. B. Ronne, “End-to-end verifiable quadratic voting with everlasting privacy,” 2019, https://fc19.ifca.ai/voting/papers/PR19.pdf.
[25] J. Camenisch, “Group signature schemes and payment systems based on the discrete logarithm problem,” 1998, https://crypto.ethz.ch/publications/files/Cameni98.pdf.
[26] A. Benhaim, B. H. Falk, and G. Tsoukalas, “Balancing power in decentralized governance: Quadratic voting and information aggregation,” 2025, https://doi.org/10.1287/mnsc.2024.08469.
[27] T. Chu and N. Laoutaris, “Fedqv: Leveraging quadratic voting in federated learning,” in *Proceedings of the ACM on Measurement and Analysis of Computing Systems 8(2)*, 2024, pp. 1–36, https://doi.org/10.1145/3656006.
[28] D. Boneh, “The decision diffie-hellman problem,” 1998, https://crypto.stanford.edu/~dabo/papers/DDH.pdf.
[29] T. ElGamal, “A public-key cryptosystem and a signature scheme based on discrete logarithms,” in *IEEE Transactions on Information Theory. 31 (4)*, 1985, pp. 469—-472, https://caislab.kaist.ac.kr/lecture/2010/spring/cs548/basic/B02.pdf.
[30] Y. Tsiounis and M. Yung, “On the security of elgamal based encryption,” in *Public Key Cryptography*, 1998, pp. 117—-134, http://dx.doi.org/10.1007/BFb0054019.
[31] A. Fiat and A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” in *CRYPTO’ 86*, 1986, pp. 186–194.
[32] M. Bellare and P. Rogaway, “Random oracles are practical: A paradigm for designing efficient protocols,” in *CCS ’93*, 1993, pp. 62–73.
[33] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, “Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts,” 2019, https://arxiv.org/pdf/1804.05141.
[34] D. Wikström, “Universally composable dkg with linear number of exponentiations,” in *Security in Communication Networks*, 2005, pp. 263–277, https://www.csc.kth.se/~dog/research/papers/Wik04bConf.pdf.
[35] “Core catalyst governance engine and utilities,” https://github.com/input-output-hk/catalyst-core.
[36] S. Goldwasser, S. Micali, and C. Rackof, “The knowledge complexity of interactive proof-system,” 1985, https://people.csail.mit.edu/silvio/Selected%20Scientific%20Papers/Proof%20Systems/The_Knowledge_Complexity_Of_Interactive_Proof_Systems.pdf.
[37] B. Bunz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, “Bulletproofs: Short proofs for confidential transactions and more,” 2017, https://eprint.iacr.org/2017/1066.pdf.
[38] S. Bayer and J. Groth, “Efficient zero-knowledge argument for correctness of a shuffle,” 2012, http://www0.cs.ucl.ac.uk/staff/J.Groth/MinimalShuffle.pdf.
[39] N. Wang, S. C.-K. Chau, and D. Liu, “Swiftrange: A short and efficient zero-knowledge range argument for confidential transactions and more,” 2023, https://eprint.iacr.org/2023/1185.pdf.
[40] “A pure-rust implementation of bulletproofs using ristretto,” https://github.com/dalek-cryptography/bulletproofs.

VI. Biography Section

Lyudmila Kovalchuk received a Dr. Habil degree in cryptology at Kyiv National University of Ukraine “Kyiv Polytechnic Institute”, Ukraine. Currently, she is a professor at G. E. Pukhov Institute for Modelling in Energy Engineering, Kyiv, Ukraine. She is a member of the state technical council for the approval of the Ukrainian national cryptographic standards and the author of the projects of National Standards. Her research interests are design and analysis of symmetric and asymmetric cryptographic primitives and random number generators, cryptocurrencies: formal analysis of proof-of-work and proof-of-stake solutions, zero-knowledge in blockchain, SNARKs.

Bingsheng Zhang received the Ph.D. degree from Worcester Polytechnic Institute, Worcester, UK. He is currently a Professor in Cybersecurity at The State Key Laboratory of Blockchain and Data Security, Zhejiang University; IOG Research Fellow. His research interests are blockchain, multi-party computation, data security, cryptography, verifiable computing, and zero-knowledge proofs.

Andrii Nastenko received a Ph.D. degree in symmetric cryptology at Kharkiv National University of Radio Electronics, Ukraine. Currently, he is a lecturer at the Department of Security of Information Technologies at Kharkiv National University of Radio Electronics, Ukraine. His research interests are blockchain technologies, zero-knowledge in blockchain, SNARKs, design and analysis of cryptographic primitives.

Zeyuan Yin received his B.E. degree in Computer Science from Shandong University in 2021. Currently, he is pursuing his Ph.D. degree at Zhejiang University. His research interests mainly focus on electronic voting (e-voting), blockchain, and zero-knowledge proofs. So far, he has published a paper on IEEE TIFS, and he won the third prize in the WPPCC 2022 Privacy-Preserving Computing Competition and the "Outstanding Team Award" in the Xinghe Cup 2023 Privacy-Preserving Computing Competition.

Roman Oliynykov received a Dr. Habil degree in symmetric cryptology at Kharkiv National University of Radio Electronics, Ukraine. Currently, he is a professor at the Department of Cybersecurity of Information Systems, Networks and Technologies at V. N. Karazin Kharkiv National University, Ukraine; visiting professor at Information Technologies Security Department at Kharkiv National University of Radio Electronics, Ukraine. His professional experience includes developing Ukrainian cryptographic standards, including the Kalyna block cipher and the Kupyna hash function. His research interests are blockchain technologies, design and analysis of cryptographic primitives, software security, and computer network security.

Mariia Rodinko received a Ph.D. in Computer Science at V. N. Karazin Kharkiv National University, Ukraine. Currently, she is an Associate Professor at the Department of Intelligent Software Systems and Technologies, V. N. Karazin Kharkiv National University, Ukraine. Her research interests are consensus algorithms and self-funded cryptocurrency systems, block cipher design, and analysis.

APPENDIX A
ZERO KNOWLEDGE PROOFS/ARGUMENTS

Key components of our protocol are Zero-Knowledge Proofs (ZKPs), also known as Σ-protocols, and Arguments of Knowledge (AoKs). These are interactive protocols in which a *prover* convinces a *verifier* that some statement is true without revealing any additional information beyond the validity of the statement [36]. Below, we introduce the core related notions used in our design.

Σ-protocol. Let $\mathcal{R}$ be a relation, $\mathcal{S}$ be a public statement, and let w be a private witness such that $(\mathcal{S}, w) \in \mathcal{R}$. A *$\Sigma$-protocol* consists of two interactive PPT algorithms $\mathcal{P}$ and $\mathcal{V}$ interacting in 3 moves:

1) $\mathcal{P}_1(\mathcal{S}, w)$: sends an initial message a to $\mathcal{V}$;
2) $\mathcal{V}_1(\mathcal{S})$: samples a random challenge e and sends it to $\mathcal{P}$;
3) $\mathcal{P}_2(\mathcal{S}, w, e)$: creates response z and sends it to $\mathcal{V}$.
   Then $\mathcal{V}_2(\mathcal{S}, a, e, z)$ outputs 1 if accepting the transcript (a, e, z) and 0 if rejecting.

There are three properties which Σ-*protocol* must have:

– ***Perfect completeness***. An honest prover having a witness w s.t. $(\mathcal{S}, w) \in \mathcal{R}$ can always convince an honest verifier:

$$Pr[a \leftarrow \mathcal{P}_1(\mathcal{S}, w); e \leftarrow \mathcal{V}_1(\mathcal{S}); z \leftarrow \mathcal{P}_2(\mathcal{S}, w, e) :$$

$$\mathcal{V}_2(\mathcal{S}, a, e, z) = 1] = 1.$$

– ***n-Special soundness***. There exists a PPT extraction algorithm $\mathcal{X}$ that can compute the witness w for the statement $\mathcal{S}$ given n accepting transcripts with the same initial message a:

$$Pr[\{(a, e_1, z_1), .., (a, e_n, z_n)\} \leftarrow \langle \mathcal{P}(\mathcal{S}, w), \mathcal{V}(\mathcal{S}) \rangle :$$

$$w \leftarrow \mathcal{X}((a, e_1, z_1), .., (a, e_n, z_n))] \geq 1 - \mathsf{negl}(\lambda),$$

where $e_1, .., e_n$ are all distinct and $\mathcal{V}(\mathcal{S}, a, e_i, z_i) = 1$ for all $i \in [n]$.
Perfect n-special soundness means that the probability of extracting the witness is exactly 1.

– ***Special honest verifier zero-knowledge (SHVZK)***. There exists a PPT simulator Sim that, given the challenge e, can generate an accepting transcript (a, e, z) for the statement $\mathcal{S}$, that is computationally indistinguishable by PPT adversary $\mathcal{A}$ from transcripts generated by honest prover and verifier:

$$Pr\left[(a, e, z) \leftarrow \langle \mathcal{P}(\mathcal{S}, w), \mathcal{V}(\mathcal{S}) \rangle : \mathcal{A}(a, e, z) = 1\right] \approx$$

$$\approx Pr\left[(a, e, z) \leftarrow Sim(\mathcal{S}, e) : \mathcal{A}(a, e, z) = 1\right].$$

Perfect SHVZK means that these two probabilities are exactly equal to each other.

Argument of knowledge (AoK). According to the *Definition 8* in [37], the notion of an *argument of knowledge* refers to a protocol that satisfies both *perfect completeness* and *computational witness extended emulation*, which we define as follows.

Computational Witness Extended Emulation. Based on *Definition 10* in [37], a protocol has computational witness extended emulation if there exists a PPT algorithm (called a *witness extended emulator*) $\mathcal{X}$ that having black-box rewindable access to any (possibly malicious) prover $\mathcal{P}^*$: (i) produces an accepting transcript that is computationally indistinguishable from a real protocol execution; (ii) with overwhelming probability outputs a witness w such that $(\mathcal{S}, w) \in \mathcal{R}$. This property implies knowledge-soundness [37], [38].

Non-Interactive Zero Knowledge Proofs/Arguments. To improve the efficiency of our protocol, we use non-interactive versions of ZKPs/AoKs obtained via the Fiat-Shamir transform [31] in the random oracle model [32] - NIZKs^{RO}, consisting of the following PPT algorithms:

– $\mathsf{Prove}(\mathcal{S}, w)$: output proof π of $(\mathcal{S}, w) \in \mathcal{R}$.
– $\mathsf{Verify}(\mathcal{S}, \pi)$: output 1 if accepting the proof π and 0 if rejecting the proof.
– $\mathsf{Sim}(\mathcal{S}, e)$: output an accepting proof π for the given statement $\mathcal{S}$ and challenge e.

According to this structure of NIZK^{RO} we define the non-interactive unit vector proof NIZK_{UV}^{RO} in Alg. 13, the exponentiaton and re-encryption correctness proof NIZK_{Exp}^{RO} in Alg. 14 and the range proof NIZK_{RP}^{RO} in Alg. 15. For brevity, we omit the RO superscript in the following usages of these notations when the context is clear.

In our definitions of the NIZKs used in this work, we may refer to interactive descriptions of the corresponding ZKPs/AoKs, assuming that the transformation into a non-interactive form is straightforward: interactive moves are easily combined into a single algorithm $\mathsf{Prove}(\cdot)$ by applying a cryptographic hash function to the statement concatenated with the prover's previous messages to deterministically derive the challenge e.

Unit vector NIZK. In our voting scheme we use NIZK_{UV} [5] to prove that a v-sized vector of ElGamal ciphertexts $\mathbf{w}^{(v)}$ is an encryption of a unit vector $\mathbf{e}^{(v)}$ under some randmoness $\mathbf{r}^{(v)}$ and the public key pk, i.e.:

$$(w_1, \ldots, w_v) =$$

$$= (\mathsf{Enc}_{\mathsf{pk}}(e_1; r_1), \ldots, \mathsf{Enc}_{\mathsf{pk}}(e_v; r_v)) \wedge (e_1, \ldots, e_v)$$

is a unit vector.

Unit Vector AoK has *perfect completeness* and is *SHVZK* according to *Theorem 2* in [5] and it has *computational witness extended emulation* as we prove in Appendix C.

Algorithm 13 Unit Vector NIZK algorithms NIZK_{UV}^{RO}

Statement, $\mathcal{S}$: group generator g, the commitment key ck, the public key pk, unit vector size v and an encrypted unit vector $\mathbf{w}^{(v)} = (\mathsf{Enc}_{\mathsf{pk}}(e_1; r_1), \ldots, \mathsf{Enc}_{\mathsf{pk}}(e_v; r_v))$.

Witness, w: unit vector $\mathbf{e}^{(v)} \in \{0,1\}^v$, encryption randomness $\mathbf{r}^{(v)} = (r_1, \ldots, r_v)$.

Algorithms:

- $\mathsf{Prove}(\mathcal{S}, w) = \mathsf{Prove}((g, \mathsf{ck}, \mathsf{pk}, v, \mathbf{w}^{(v)}), (\mathbf{e}^{(v)}, \mathbf{r}^{(v)}))$: output a proof π_{UV} created according to the proving algorithm in Fig.12 of [5].
- $\mathsf{Verify}(\mathcal{S}, \pi) = \mathsf{Verify}((g, \mathsf{ck}, \mathsf{pk}, v, \mathbf{w}^{(v)}), \pi_{UV})$: output 1 if the verification algorithm in Fig.12 of [5] accepts the proof π_{UV}, otherwise output 0.
- $\mathsf{Sim}(\mathcal{S}, e) = \mathsf{Sim}((g, \mathsf{ck}, \mathsf{pk}, v, \mathbf{w}^{(v)}), e)$: output (π_{UV}, a), where π_{UV} is a proof and a is an initial message both created according to the Sim algorithm in the proof of *Theorem 2* in [5].

Exponentiation and re-encryption correctness NIZK. We use NIZK_{Exp} to prove that the vector of ElGamal ciphertexts $\mathbf{W}^{(v)}$ of length v is obtained by element-wise exponentiation of another vector $\mathbf{w}^{(v)}$ to the power s, followed by re-encryption under the public key pk using randomness $\mathbf{t}^{(v)}$, while the exponent s is perfectly hidden by a Pedersen commitment C, computed with some randomness r:

$$(W_1, \ldots, W_v) =$$
$$= (w_1^s \cdot \mathsf{Enc}_{\mathsf{pk}}(0, t_1), \ldots, w_v^s \cdot \mathsf{Enc}_{\mathsf{pk}}(0, t_v)) \wedge C = g^s \cdot \mathsf{ck}^r.$$

Exponentiation and re-encryption correctness: AoK has *perfect completeness, computational witness extended emulation, and is SHVZK*, as we prove in Appendix B.

Algorithm 14 Exponentiation and re-encryption correctness NIZK algorithms NIZK_{Exp}^{RO}

Statement, $\mathcal{S}$: group generator g, the commitment key ck, the encryption key pk, unit vector size v, two vectors of ElGamal ciphertexts $\mathbf{w}^{(v)}$ and $\mathbf{W}^{(v)}$, and a Pedersen commitment $C = g^s \cdot \mathsf{ck}^r$ of some exponent s such that $(W_1, \ldots, W_v) = (w_1^s \cdot \mathsf{Enc}_{\mathsf{pk}}(0, t_1), \ldots, w_v^s \cdot \mathsf{Enc}_{\mathsf{pk}}(0, t_v))$.

Witness, w: exponent s, commitment randomness r and re-encryption randomness vector $\mathbf{t}^{(v)}$.

Algorithms:

- $\mathsf{Prove}(\mathcal{S}, w) = \mathsf{Prove}((g, \mathsf{ck}, \mathsf{pk}, v, \mathbf{w}^{(v)}, \mathbf{W}^{(v)}, C), (s, r, \mathbf{t}^{(v)}))$: output a proof π_{Exp} created according to the proving algorithm (Alg. 16).
- $\mathsf{Verify}(\mathcal{S}, \pi) = \mathsf{Verify}((g, \mathsf{ck}, \mathsf{pk}, v, \mathbf{w}^{(v)}, \mathbf{W}^{(v)}, C), \pi_{Exp})$: output 1 if the verification algorithm (Alg. 16) accepts the proof π_{Exp}, otherwise output 0.
- $\mathsf{Sim}(\mathcal{S}, e) = \mathsf{Sim}((g, \mathsf{ck}, \mathsf{pk}, v, \mathbf{w}^{(v)}, \mathbf{W}^{(v)}, C), e)$: output (π_{Exp}, a), where π_{Exp} is a proof and a is an initial message both created according to the Sim algorithm in the ZK-property proof of *Theorem 2*.

Range proof NIZK. We use NIZK_{RP} to prove that $C \in \mathbb{G}$ is the Pedersen commitment of a non-negative value s under some randomness r i.e.:

$$C = g^s \cdot \mathsf{ck}^r \wedge s \in \left[0,\ 2^N - 1\right], \text{ where } N > 0.$$

Any existing range proof NIZK that is consistent with the statement and witness definitions in Alg. 15 and has perfect completeness, knowledge-soundness, and SHVZK properties is suitable for our scheme (for example, Bulletproofs [37] or Swiftrange [39]).

In this work, we define NIZK_{RP} for the Bulletproofs AoK [37], which we use in our prototype implementation due to the public availability of its efficient implementation in the Rust language [40]. This AoK has *perfect completeness, computational witness extended emulation* and is *perfect SHVZK* according to *Theorem 3* in [37].

Algorithm 15 Range proof NIZK algorithms NIZK_{RP}^{RO}

Statement, $\mathcal{S}$: group generator g, the commitment key ck, the length in bits of the committed value N and the Pedersen commitment $C = g^s \cdot \mathsf{ck}^r$ of some value $s \in \left[0,\ 2^N - 1\right]$.

Witness, w: committed value s and commitment randomness r.

Algorithms:

- $\mathsf{Prove}(\mathcal{S}, w) = \mathsf{Prove}((g, \mathsf{ck}, N, C), (s, r))$: output a proof π_{RP} created according to the proving algorithm in Section 4.1 of [37].
- $\mathsf{Verify}(\mathcal{S}, \pi) = \mathsf{Verify}((g, \mathsf{ck}, N, C), \pi_{RP})$: output 1 if the verification algorithm in Section 4.1 of [37] accepts the proof π_{RP}, otherwise output 0.
- $\mathsf{Sim}(\mathcal{S}, e) = \mathsf{Sim}((g, \mathsf{ck}, N, C), e)$: output (π_{RP}, a), where π_{RP} is a proof and a is an initial message both created according to the simulator's algorithm in the proof of *Theorem 3* in [37].

APPENDIX B
EXPONENTIATION AND RE-ENCRYPTION CORRECTNESS AOK

Algorithm 16 Exponentiation and Re-encryption Correctness argument of knowledge

Statement: For a given $g \in \mathbb{G}$, $h \in \mathbb{G}$, $\mathsf{pk} \in \mathbb{G}$, a Pedersen commitment $C \in \mathbb{G}$ and two vectors of ElGamal ciphertexts $\mathbf{w}^{(k)}$ and $\mathbf{W}^{(k)}$ prove that: $W_i = w_i^s \cdot \mathsf{Enc}_{\mathsf{pk}}(0, t_i)$ for $i \in [k] \wedge C = g^s \cdot h^r$.

Witness: $s \in \mathbb{Z}_q, r \in \mathbb{Z}_q$ and $t_1, \ldots t_k \in \mathbb{Z}_q$

Protocol:

1) *Verifier:*
   - samples random challenge $\rho \leftarrow \mathbb{Z}_q$ and sends it to *Prover*; (for NIZK: $\rho = \mathtt{Hash}((g, h, \mathsf{pk}, \mathbf{w}^{(k)}, \mathbf{W}^{(k)}, C))$.
2) *Prover*:
   - selects random $\alpha \leftarrow \mathbb{Z}_q, \beta \leftarrow \mathbb{Z}_q$ and $\gamma \leftarrow \mathbb{Z}_q$;
   - computes: $\widetilde{w} = \prod_{i=1}^k w_i^{\rho^{i-1}}$, $\widetilde{W'} = \widetilde{w}^{\alpha} \cdot \mathsf{Enc}_{\mathsf{pk}}(0, \gamma)$ and $C' = g^{\alpha} h^{\beta}$;
   - sends $(C', \widetilde{W'})$ to *Verifier*.
3) *Verifier:*
   - samples random challenge $e \leftarrow \mathbb{Z}_q$ and sends it to *Prover*; (for NIZK: $e = \mathtt{Hash}((g, h, \mathsf{pk}, \mathbf{w}^{(k)}, \mathbf{W}^{(k)}, C) \,||\, (C', \widetilde{W'}))$).
4) *Prover*:
   - computes: $z = s \cdot e + \alpha, y = r \cdot e + \beta, x = e \cdot \sum_{i=1}^k t_i \rho^{i-1} + \gamma$;
   - sends (z, y, x) to *Verifier*.

Verification:
(for NIZK: *Verifier* derives ρ and e as above)

- Computes: $\widetilde{w} = \prod_{i=1}^k w_i^{\rho^{i-1}}$ and $\widetilde{W} = \prod_{i=1}^k W_i^{\rho^{i-1}}$;
- Checks the next equalities:
  - $C^e \cdot C' = g^z h^y$;
  - $\widetilde{W}^e \cdot \widetilde{W'} = \widetilde{w}^z \cdot \mathsf{Enc}_{\mathsf{pk}}(0, x)$.

Theorem 2. *The protocol is a special honest-verifier zero-knowledge (SHVZK) argument of knowledge of $s, r, t_1, \ldots, t_k$ such that for a Pedersen commitment $C \in \mathbb{G}$ and two vectors of ElGamal ciphertexts $\mathbf{w}^{(k)}$ and $\mathbf{W}^{(k)}$: $W_i = w_i^s \cdot \mathsf{Enc}_{\mathsf{pk}}(0, t_i)$ for $i \in [k] \wedge C = g^s \cdot h^r$.*

Proof. **Perfect Completeness**. It may be easily checked that:

$$C^e \cdot C' = (g^s h^r)^e \cdot g^\alpha h^\beta = g^{se+\alpha} \cdot h^{re+\beta} = g^z \cdot h^y;$$

$$\begin{aligned}
\widetilde{W}^e \cdot \widetilde{W}' &= \left(\prod_{i=1}^{k} (w_i^s \cdot \mathsf{Enc}_{\mathsf{pk}}(0, t_i))^{\rho^{i-1}}\right)^e \cdot \widetilde{w}^\alpha \cdot \mathsf{Enc}_{\mathsf{pk}}(0, \gamma) \\
&= \left(\prod_{i=1}^{k} w_i^{s\rho^{i-1}}\right)^e \cdot \mathsf{Enc}_{\mathsf{pk}}\left(0, e\sum_{i=1}^{k} t_i \rho^{i-1}\right) \\
&\quad \cdot \widetilde{w}^\alpha \cdot \mathsf{Enc}_{\mathsf{pk}}(0, \gamma) \\
&= \widetilde{w}^{se+\alpha} \cdot \mathsf{Enc}_{\mathsf{pk}}\left(0, e\sum_{i=1}^{k} t_i \rho^{i-1} + \gamma\right) \\
&= \widetilde{w}^z \cdot \mathsf{Enc}_{\mathsf{pk}}(0, x).
\end{aligned}$$

Computational Witness Extended Emulation. There exists a PPT witness extended emulator $\mathcal{X}$ which runs $\langle Prover, Verifier\rangle$ to get a transcript. If $Verifier$ rejects the transcript, then $\mathcal{X}$ does not need to do anything else. If $Verifier$ accepts the transcript, $\mathcal{X}$ will extract the witnesses $s, r, t_1, \ldots, t_k$ by rewinding the $Prover$ and running it with fresh challenges ρ and e until it has $2k$ accepting transcripts. More specifically, the emulator $\mathcal{X}$ operates in two stages:

- Rewinds the $Prover$ to the first challenge phase (ρ) and runs it with fresh challenges ρ and e to obtain k acceptable transcripts $\{(\rho_j, C_j', \widetilde{W}_j', e_j, z_j, y_j, x_j)\}_{j=1}^k$.
- For each of k obtained transcripts rewinds the $Prover$ to the second challenge phase (e) and runs it with fresh challenge e_j'.

As a result the emulator $\mathcal{X}$ obtains k pairs of accepting transcripts with a common prefix $a_j = (\rho_j, C_j', \widetilde{W}_j')$: $\{((a_j, e_j, z_j, y_j, x_j), (a_j, e_j', z_j', y_j', x_j'))\}_{j=1}^k$. From these pairs of transcripts, the emulator $\mathcal{X}$ extracts $T_j = \sum_{i=1}^k t_i \rho_j^{i-1}$ for $j \in [k]$ and the witnesses s and r (s, r can be extracted from any pair of these transcripts) by computing:

$$s = \frac{z_j' - z_j}{e_j' - e_j};\ r = \frac{y_j' - y_j}{e_j' - e_j};\ T_j = \frac{x_j' - x_j}{e_j' - e_j}.$$

Now the emulator $\mathcal{X}$ has k pairs $\{(\rho_j, T_j)\}_{j=1}^k$ where T_j is the value in point ρ_j of the degree-$(k-1)$ polynomial $f(\rho) = \sum_{i=1}^k t_i \rho^{i-1}$. The emulator can recover the witnesses $t_1, \ldots, t_k$ by solving the following linear system defined by the Vandermonde matrix $V \in \mathbb{Z}_q^{k \times k}$ over the distinct challenge points ρ_j:

$$\begin{bmatrix} 1 & \rho_1 & \rho_1^2 & \cdots & \rho_1^{k-1} \\ 1 & \rho_2 & \rho_2^2 & \cdots & \rho_2^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \rho_k & \rho_k^2 & \cdots & \rho_k^{k-1} \end{bmatrix} \cdot \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_k \end{bmatrix} = \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_k \end{bmatrix}$$

Since the values $\rho_1, \ldots, \rho_k$ are all distinct, the Vandermonde matrix V is invertible. Therefore, the emulator $\mathcal{X}$ can uniquely and efficiently solve this system to recover all witnesses $t_1, \ldots, t_k$.

Finally, by the Schwartz–Zippel lemma, the probability that a prover without knowledge of valid $t_1, \ldots, t_k$, can construct a different polynomial $f'(\rho)$ such that $f'(\rho) = f(\rho)$ at a randomly sampled point ρ is at most $\frac{k-1}{q}$.

Zero-Knowledge. In terms of special honest verifier zero-knowledge, we construct a simulator Sim, that given the statement $(g, h, \mathsf{pk}, \mathbf{w}^{(k)}, \mathbf{W}^{(k)}, C)$ and the challenges (ρ, e) as inputs proceeds as follows:

- computes $\widetilde{w} = \prod_{i=1}^k w_i^{\rho^{i-1}}$ and $\widetilde{W} = \prod_{i=1}^k W_i^{\rho^{i-1}}$;
- samples random $z \leftarrow \mathbb{Z}_q,\ y \leftarrow \mathbb{Z}_q,\ x \leftarrow \mathbb{Z}_q$;
- computes: $C' = g^z h^y C^{-e}$, $\widetilde{W}' = \widetilde{w}^z \cdot \mathsf{Enc}_{\mathsf{pk}}(0, x) \cdot \widetilde{W}^{-e}$.

The simulated transcript $(\rho, C', \widetilde{W}', e, z, y, x)$ is indistinguishable from the real one since z, y, x, C', $\widetilde{W}'$ are random in both transcripts. □

APPENDIX C
COMPUTATIONAL WITNESS EXTENDED EMULATION FOR UNIT VECTOR AOK

In this section, we provide a proof of computational witness extended emulation property for the Unit Vector AoK described in Figure 12 of [5].

Computational Witness Extended Emulation. There exists a PPT witness extended emulator $\mathcal{X}$ which runs $\langle Prover, Verifier\rangle$ to get a transcript. If $Verifier$ rejects the transcript, then $\mathcal{X}$ does not need to do anything else. If $Verifier$ accepts the transcript, $\mathcal{X}$ will extract the witnesses $i_0, \ldots, i_{\log n - 1}$ and $r_0, \ldots, r_{n-1}$ by rewinding the $Prover$ and running it with fresh challenges y and x until it has $n(\log n + 1)$ accepting transcripts. More specifically, the emulator $\mathcal{X}$ operates in two stages:

- Rewinds the $Prover$ to the first challenge phase (y) and runs it with fresh challenges y and x to obtain n accepting transcripts with common prefix $a = \{I_l, B_l, A_l\}_{\ell=1}^{\log n}$:

$$\left\{\left(a, y_h, \{D_{\ell h}\}_{\ell=0}^{\log n - 1}, x_h, R_h, \{z_{\ell h}, w_{\ell h}, v_{\ell h}\}_{\ell=1}^{\log n}\right)\right\}_{h=1}^{n}.$$

- For each of n obtained transcripts rewinds the $Prover$ to the second challenge phase (x) and runs it with fresh challenges $\{x_{hk}\}_{k=1}^{\log n}$ to obtain $\log n + 1$ accepting transcripts for the same challenge y_h.

As a result the emulator $\mathcal{X}$ obtains n tuples of $\log n + 1$ accepting transcripts with a common prefix $b_h = (a, y_h, \{D_{\ell h}\}_{\ell=0}^{\log n - 1})$:

$$\left\{\left\{\left(b_h, x_{hk}, R_{hk}, \{z_{\ell hk}, w_{\ell hk}, v_{\ell hk}\}_{\ell=1}^{\log n}\right)\right\}_{k=0}^{\log n}\right\}_{h=1}^{n}.$$

Emulator $\mathcal{X}$ can extract the witnesses $i_1, \ldots, i_{\log n}$ using any 2 transcripts taken from any of these tuples by computing:

$$i_l = \frac{z_{\ell h' k'} - z_{\ell h k}}{x_{h' k'} - x_{h k}}.$$

Further, from these $(\log n + 1)$-sized tuples of transcripts the emulator $\mathcal{X}$ computes the values $\{R'_{h \log n}\}_{h=1}^n$ which are the

leading coefficients of the polynomials $f_h(x) = \sum_{l=0}^{\log n} R'_{hl} x^l$ by solving n linear systems defined by the Vandermonde matrices V_h each over the distinct challenge points x_{hk}:

$$\begin{bmatrix} 1 & x_{h0} & x_{h0}^2 & \cdots & x_{h0}^{\log n} \\ 1 & x_{h1} & x_{h1}^2 & \cdots & x_{h1}^{\log n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{h\log n} & x_{h\log n}^2 & \cdots & x_{h\log n}^{\log n} \end{bmatrix} \cdot \begin{bmatrix} R'_{h0} \\ R'_{h1} \\ \vdots \\ R'_{h\log n} \end{bmatrix} = \begin{bmatrix} R_{h0} \\ R_{h1} \\ \vdots \\ R_{h\log n} \end{bmatrix}$$

Since the values $x_{h0}, \ldots, x_{h\log n}$ are all distinct, each Vandermonde matrix V_h is invertible. Therefore, the emulator $\mathcal{X}$ can solve each of h systems to recover the values $\{R'_{h\log n}\}_{h=1}^{n}$.

Now the emulator $\mathcal{X}$ has n pairs $\{(y_h, R'_{h\log n})\}_{h=1}^{n}$ where $R'_{h\log n}$ is the value in point y_h of the degree-$(n-1)$ polynomial $g(y) = \sum_{j=0}^{n-1} r_j \cdot y^j$. The emulator can recover the witnesses $r_0, \ldots, r_{n-1}$ by solving the linear system defined by the Vandermonde matrix W over the distinct challenge points y_h:

$$\begin{bmatrix} 1 & y_1 & y_1^2 & \cdots & y_1^{n-1} \\ 1 & y_2 & y_2^2 & \cdots & y_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & y_n & y_n^2 & \cdots & y_n^{n-1} \end{bmatrix} \cdot \begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_{n-1} \end{bmatrix} = \begin{bmatrix} R'_{0\log n} \\ R'_{1\log n} \\ \vdots \\ R'_{n\log n} \end{bmatrix}$$

Since the values $y_1, \ldots, y_n$ are all distinct, the Vandermonde matrix W is invertible. Therefore, the emulator $\mathcal{X}$ can solve this system to recover all witnesses $r_0, \ldots, r_{n-1}$.

APPENDIX D
PROOF OF THEOREM 1

Proof of Theorem 1. 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 $\mathsf{EXEC}^{\mathcal{F}_{\text{BLOCKCHAIN}}, \mathcal{F}_{\text{DKG}}^{t,u}}_{\Pi_{\text{VOTE}}^{t,u,m,n}, \mathcal{A}, \mathcal{Z}}$ where the parties $\mathrm{V} = \{V^{(1)}, \ldots, V^{(n)}\}$, $\mathrm{D} = \{D^{(1)}, \ldots, D^{(m)}\}$ and $\mathrm{T} = \{T^{(1)}, \ldots, T^{(u)}\}$ run protocol $\Pi_{\text{VOTE}}^{t,u,m,n}$ in the $\{\mathcal{F}_{\text{BLOCKCHAIN}}, \mathcal{F}_{\text{DKG}}^{t,u}\}$-hybrid world and the corrupted parties are controlled a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\mathsf{EXEC}_{\mathcal{F}_{\text{VOTE}}^{t,u,m,n}, \mathcal{S}, \mathcal{Z}}$ where the parties interact with functionality $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ in the ideal world and corrupted parties are controlled by the simulator $\mathcal{S}$.

Let $\mathrm{V}_{corr} \subseteq \mathrm{V}$, $\mathrm{D}_{corr} \subseteq \mathrm{D}$, and $\mathrm{T}_{corr} \subseteq \mathrm{T}$ be the set of corrupted voters, delegates, and trustees, respectively. Note that under the DDH assumption, the underlying Lifted ElGamal encryption scheme is IND-CPA secure, and the employed NIZK proofs, derived via the Fiat–Shamir transformation from σ-protocols, satisfy perfect completeness, n-special soundness, and SHVZK in the random oracle model.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$ and simulating honest voters $V^{(i)} \in \mathrm{V} \setminus \mathrm{V}_{corr}$, honest delegates $D^{(i)} \in \mathrm{D} \setminus \mathrm{D}_{corr}$, honest trustees $T^{(i)} \in \mathrm{T} \setminus \mathrm{T}_{corr}$ and functionalities $\mathcal{F}_{\text{BLOCKCHAIN}}, \mathcal{F}_{\text{DKG}}^{t,u}$. $\mathcal{S}$ uses the supporting algorithm $\mathsf{SimBallot}$ for honest voters'/delegates' ballots simulation and $\mathsf{SimProof}$ for NIZK proofs simulation, defined in Alg. 17 and Alg. 18, respectively. In addition, $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

In the preparation phase:

- Upon receiving $(\text{INITNOTIFY}, sid, T^{(i)})$ from the external $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ for an honest voting committee $T^{(i)} \in \mathrm{T} \setminus \mathrm{T}_{corr}$, the simulator $\mathcal{S}$ acts as $T^{(i)}$, following the protocol $\Pi_{\text{VOTE}}^{t,u,m,n}$ as if $T^{(i)}$ receives (INIT, sid) from the environment $\mathcal{Z}$.
- $\mathcal{S}$ simulates $\mathcal{F}_{\text{DKG}}^{t,u}$ so that it generates and stores $(\mathsf{pk}, \mathsf{sk})$.

In the voting/delegation phase:

- Upon receiving $(\text{VOTENOTIFY}, sid, D^{(i)})$ from the external $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ for an honest delegate $D^{(i)} \in \mathrm{D} \setminus \mathrm{D}_{corr}$ the simulator $\mathcal{S}$ computes the simulated delegate's ballot $B_D^{(i)} = \mathsf{SimBallot}_D(\mu,\ l,\ N, \mathsf{pk}, \mathsf{ck})$ and posts it to $\mathcal{F}_{\text{BLOCKCHAIN}}$.
- Upon receiving $(\text{CASTNOTIFY}, sid, V^{(i)})$ from the external $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ for an honest voter $V^{(i)} \in \mathrm{V} \setminus \mathrm{V}_{corr}$ the simulator $\mathcal{S}$ computes the simulated voter's ballot $B_V^{(i)} = \mathsf{SimBallot}_V(s^{(i)},\ l,\ N, \mathsf{pk}, \mathsf{ck})$ and posts it to $\mathcal{F}_{\text{BLOCKCHAIN}}$.
- Once the simulated $\mathcal{F}_{\text{BLOCKCHAIN}}$ receives $(\text{POST}, sid, B_D^{(i)})$ from a corrupted delegate $D^{(i)} \in \mathrm{D}_{corr}$, the simulator $\mathcal{S}$ validates the $D^{(i)}$'s ballot: $valid := \mathsf{VerifyBallot}_{\mathrm{D}}(B_D^{(i)}, \mu, N, \mathsf{pk}, \mathsf{ck})$. If $valid = 0$, $\mathcal{S}$ ignores the request; otherwise it parses $B_D^{(i)}$ as $\left(\left\{\left(\mathbf{w}_j^{(3)}, \mathbf{W}_j^{(3)}, C_j, \pi_j^{\mathsf{UV}}, \pi_j^{\mathsf{Exp}}, \pi_j^{\mathsf{RP}}\right)\right\}_{j=1}^{l},\ t_{\mathsf{sum}}\right)$ and uses sk to decrypt $\mathbf{W}_j^{(3)}$, obtaining the vote v_i. Then $\mathcal{S}$ sends (VOTE, sid, v_i) to $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ on behalf of $D^{(i)}$.
- Once the simulated $\mathcal{F}_{\text{BLOCKCHAIN}}$ receives $(\text{POST}, sid, B_V^{(i)})$ from a corrupted voter $V^{(i)} \in \mathrm{V}_{corr}$, the simulator $\mathcal{S}$ validates the $V^{(i)}$'s ballot: $valid := \mathsf{VerifyBallot}_{\mathrm{V}}(B_V^{(i)}, s^{(i)}, N, \mathsf{pk}, \mathsf{ck})$. If $valid = 0$, $\mathcal{S}$ ignores the request; otherwise it parses $B_V^{(i)}$ as $\left(\left\{\left(\mathbf{w}_j^{(m+3)}, \mathbf{W}_j^{(m+3)}, C_j, \pi_j^{\mathsf{UV}}, \pi_j^{\mathsf{Exp}}, \pi_j^{\mathsf{RP}}\right)\right\}_{j=1}^{l},\ t_{\mathsf{sum}}\right)$ and uses sk to decrypt $\mathbf{W}_j^{(m+3)}$, obtaining the vote v_i. Then $\mathcal{S}$ sends (CAST, sid, v_i) to $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ on behalf of $V^{(i)}$.

In the tally phase:

- Upon receiving $(\text{LEAKDEL}, sid, \delta)$ from the external $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$, the simulator $\mathcal{S}$ simulates the honest trustees' decryption shares according to δ, and it simulates the corresponding NIZK proofs in the DELCAL processing phase.
- Upon receiving $(\text{LEAKTALLY}, sid, \tau)$ from the external $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$, the simulator $\mathcal{S}$ simulates the honest trustees' decryption shares according to τ, and it simulates the corresponding NIZK proofs in the TALLY processing phase.
- Upon receiving $(\text{DELCALNOTIFY}, sid, T^{(i)})$ from the external $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ for an honest trustee $T^{(i)} \in \mathrm{T} \setminus \mathrm{T}_{corr}$, the simulator $\mathcal{S}$ acts as $T^{(i)}$, following the protocol $\Pi_{\text{VOTE}}^{t,u,m,n}$ as if $T^{(i)}$ receives (DELCAL, sid) from $\mathcal{Z}$.
- Upon receiving $(\text{TALLYNOTIFY}, sid, T^{(i)})$ from the external $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$ for an honest trustee $T^{(i)} \in \mathrm{T} \setminus \mathrm{T}_{corr}$, the

simulator $\mathcal{S}$ acts as $T^{(i)}$, following the protocol $\Pi_{\text{VOTE}}^{t,u,m,n}$ as if $T^{(i)}$ receives (TALLY, sid) from $\mathcal{Z}$.

Algorithm 17 Ballot simulation algorithm $\mathsf{SimBallot}_{BT}$

Input: ballot type $BT \in \{\texttt{"V"}, \texttt{"D"}\}$; public total stake s_{total}, number of votes for projects l, maximal size in bits of a stake share value N; encryption public key pk; commitment key ck.
Output: simulated ballot B.

Algorithm:

- Initialize the set of encrypted votes $EV := \emptyset$ and the randomness sum $t_{\mathsf{sum}} := 0$.
- Set $\theta := m + 3$ if $BT = \texttt{"V"}$, otherwise $\theta := 3$;
- Set $\mathbf{0}^{(\theta)}$ as a θ-sized vector of all zeroes.
- Generate l random stake shares consistent with public total stake s_{total} as follows:
  - Sample random $s_1, \ldots, s_{l-1} \leftarrow \mathbb{Z}_q$.
  - Set $s_l := s_{\mathsf{total}} - \sum_{j=1}^{l-1} s_j$.
- For $j \in [l]$ do the following steps:
  - Sample randomness $\mathbf{r}_j^{(\theta)} \leftarrow \mathbb{Z}_q^{\theta}$.
  - Compute Lifted ElGamal encryption of $\mathbf{0}^{(\theta)}$: $\mathbf{w}_j^{(\theta)} := \mathsf{Enc}_{\mathsf{pk}}(\mathbf{0}^{(\theta)}, \mathbf{r}_j^{(\theta)})$.
  - Sample $\mathbf{p}_j^{(\theta)} \leftarrow \mathbb{Z}_q^{\theta}$ and compute: $\mathbf{W}_j^{(\theta)} := \mathsf{Exp}(\mathbf{w}_j^{(\theta)}, \mathbf{0}^{(\theta)}) \cdot \mathsf{Enc}_{\mathsf{pk}}(\mathbf{0}^{(\theta)}, \mathbf{p}_j^{(\theta)})$.
  - Sample $t_j \leftarrow \mathbb{Z}_q$ and update $t_{\mathsf{sum}} := t_{\mathsf{sum}} + t_j$.
  - Compute Pedersen commitment: $C_j := \mathsf{Com}_{\mathsf{ck}}(s_j, t_j)$.
  - Simulate the NIZK range proof: $\pi_j^{\mathsf{RP}} = \mathsf{SimProof}(\mathsf{NIZK}_{RP}, (g, \mathsf{ck}, N, C_j))$.
  - Simulate the NIZK proof of unit vector correctness: $\pi_j^{\mathsf{UV}} = \mathsf{SimProof}(\mathsf{NIZK}_{UV}, (g, \mathsf{ck}, \mathsf{pk}, \theta, \mathbf{w}_j^{(\theta)}))$.
  - Simulate the NIZK proof of correct exponentiation: $\pi_j^{\mathsf{Exp}} = \mathsf{SimProof}(\mathsf{NIZK}_{Exp}, (g, \mathsf{ck}, \mathsf{pk}, \theta, \mathbf{w}_j^{(\theta)}, \mathbf{W}_j^{(\theta)}, C_j))$.
  - Append to encrypted votes: $EV := EV \cup \left\{\left(\mathbf{w}_j^{(\theta)}, \mathbf{W}_j^{(\theta)}, C_j, \pi_j^{\mathsf{UV}}, \pi_j^{\mathsf{Exp}}, \pi_j^{\mathsf{RP}}\right)\right\}$.

Return: $B := (EV,\ t_{\mathsf{sum}})$.

Algorithm 18 NIZK^{RO} proof simulation algorithm $\mathsf{SimProof}$

Input: The NIZK^{RO} algorithm and statement S.
Output: simulated proof π.

Algorithm:

- Sample a random challenge $c \leftarrow \mathbb{Z}_q$.
- Run the corresponding simulation algorithm to obtain a simulated proof π and initial message a: $(\pi, a) := \mathsf{NIZK}^{RO}.\mathsf{Sim}(S, c)$.
- Check whether the random oracle has already been queried on input $(S \,\|\, a)$. If so, repeat the previous step. Otherwise, proceed *(since a is sampled at random, the simulation will succeed with overwhelming probability)*.
- Program the random oracle so that it returns c in response to input $(S \,\|\, a)$.

Return: π.

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \ldots, \mathcal{H}_4$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathsf{EXEC}^{\mathcal{F}_{\text{LEDGER}}, \mathcal{F}_{\text{DKG}}^{t,u}}_{\Pi_{\text{VOTE}}^{t,u,m,n}, \mathcal{A}, \mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except the followings. During the tally phase, $\mathcal{H}_1$ uses the NIZK simulator to simulate all the decryption proofs instead of the real ones.

Claim: $\mathcal{H}_1$ and $\mathcal{H}_0$ are indistinguishable if the underlying decryption NIZK is simulatable ZK.

Proof. The adversary's advantage is bounded by the ZK property of the decryption NIZK. □

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except the followings. During the tally phase, the honest trustees' decryption shares are backward-calculated from the δ and τ received from the $\mathcal{F}_{\text{VOTE}}^{t,u,m,n}$.

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

Proof. The distribution of the decryption shares in $\mathcal{H}_2$ has an identical distribution to the shares in $\mathcal{H}_1$. □

Hybrid $\mathcal{H}_3$: $\mathcal{H}_3$ is the same as $\mathcal{H}_2$ except the following: During the voting/delegation phase, in $\mathcal{H}_3$, the honest voter's $V^{(i)}$'s encrypted ballots are replaced with simulated ballots $B_V^{(i)} = \mathsf{SimBallot}_V(s^{(i)},\ l,\ N, \mathsf{pk}, \mathsf{ck})$.

Claim: $\mathcal{H}_3$ and $\mathcal{H}_2$ are indistinguishable if the underlying encryption scheme is IND-CPA secure, and the NIZK_{UV}, NIZK_{Exp}, and NIZK_{RP} are simulatable ZK.

Proof. The proof is straightforward. Namely, if an adversary $\mathcal{A}$ can distinguish $\mathcal{H}_3$ from $\mathcal{H}_2$, then we can construct an adversary $\mathcal{B}$ who can either break the IND-CPA game of the PKE encryption or the ZK property of NIZK_{UV} or of NIZK_{Exp} or of NIZK_{RP}. □

Hybrid $\mathcal{H}_4$: $\mathcal{H}_4$ is the same as $\mathcal{H}_3$ except that during the voting/delegation phase, in $\mathcal{H}_4$, the honest delegate $D^{(i)}$'s encrypted ballots are replaced with simulated ballots $B_D^{(i)} = \mathsf{SimBallot}_D(\mu,\ l,\ N, \mathsf{pk}, \mathsf{ck})$.

Claim: $\mathcal{H}_4$ and $\mathcal{H}_3$ are indistinguishable if the underlying encryption scheme is IND-CPA secure and the NIZK_{UV}, NIZK_{Exp}, and NIZK_{RP} are simulatable ZK.

Proof. The same as the previous proof. □

The adversary's view of $\mathcal{H}_4$ is identical to the simulated view $\mathsf{EXEC}_{\mathcal{F}_{\text{VOTE}}^{t,u,m,n}, \mathcal{S}, \mathcal{Z}}$. 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. This concludes our proof. □