A Scalable Coercion-resistant Blockchain Decision-making Scheme

Zeyuan Yin
Zhejiang University
zeyuanyin@zju.edu.cn

Bingsheng Zhang
Zhejiang University
bingsheng@zju.edu.cn

Andrii Nastenko
IOG Singapore Pte Ltd
andrii.nastenko@iohk.io

Roman Oliynykov
IOG Singapore Pte Ltd;
V.N.Karazin Kharkiv National University, Ukraine
roman.oliynykov@iohk.io

Kui Ren
Zhejiang University
kuiren@zju.edu.cn

Abstract

Typically, a decentralized collaborative blockchain decision-making mechanism is realized by remote voting. To date, a number of blockchain voting schemes have been proposed; however, to the best of our knowledge, none of these schemes achieve coercion-resistance. In particular, for most blockchain voting schemes, the randomness used by the voting client can be viewed as a witness/proof of the actual vote, which enables improper behaviors such as coercion and vote-buying. Unfortunately, the existing coercion-resistant voting schemes cannot be directly adopted in the blockchain context. In this work, we design the first scalable coercion-resistant blockchain decision-making scheme that supports private differential voting power and 1-layer liquid democracy as introduced by Zhang *et al.* (NDSS'19). Its overall complexity is $O(n)$, where n is the number of voters. Moreover, the ballot size is reduced from Zhang *et al.*'s $\Theta(m)$ to $\Theta(1)$, where m is the number of experts and/or candidates. Its incoercibility is formally proven under the UC incoercibility framework by Alwen *et al.* (Crypto'15). We implement a prototype of the scheme and the evaluation result shows that our scheme's tally execution time is more than 6x faster than VoteAgain (USENIX'20) in an election with over 10,000 voters and over 50% extra ballot rate.

1 Introduction

Blockchain technology enjoys its popularity since the invention of Bitcoin in 2008, and it continues to reshape our digital society with its property for decentralization, transparency, and security. Democracy on the blockchain has the potential to transform the way political systems function, creating a more equitable and inclusive future.

Democracy on the blockchain. In a democratic blockchain system, stakeholders have the right to participate in the decision-making process through verifiable remote voting where everyone can express his opinion. Usually, in a blockchain voting, each participant's voting power is different and is proportional to his stake, which is called differential voting power in this work. This is one of the main differences between blockchain voting and conventional elections, where typically one participant has one vote. On the other hand, direct democracy might not always be the best choice for a blockchain decision-making process. In practice, to make a wise decision, a stakeholder needs to rationally put substantial effort into informing himself, and also expert knowledge throughout the process. Therefore, letting elites lead the decision-making might be an optimization in most cases. Some systems such as ZCash [6] use a small committee (consisting of several experts) to make the decisions; however, this has the risk of centralization, i.e., if the committee behaves maliciously, there is no mechanism for stakeholders to alter their decisions whatsoever.

The concept of liquid democracy has been proposed to achieve better collaborative intelligence. Liquid democracy (also known as delegative democracy [19]) is a hybrid of direct democracy and representative democracy. It provides the benefits of both systems (whilst avoiding their drawbacks) by enabling organizations to take advantage of experts in a blockchain voting process, as well as giving the stakeholders the opportunity to vote. For each proposal, a voter can either vote directly or delegate his voting power to an expert who is knowledgeable and renowned in the corresponding area.

Zhang *et al.* [31] proposed a treasury system that supports liquid democracy. However, their scheme has the following two drawbacks: (i) the ballot size is linear in the number of candidates and/or experts; (ii) it is not coercion-resistant.

The coercion problem in remote voting. In real-world voting, a voting booth gives a voter privacy and protects him from being coerced. However, in remote voting, the voting procedure can be viewed as a probabilistic algorithm that takes as input a random coin and the voter's choice. If the output is published on the bulletin board, then we have a problem: the input and randomness used in the voting procedure can be viewed as a proof of casting a certain ballot, and anyone can run the probabilistic algorithm again to verify it, which makes coercion and vote-buying possible. Many well-known e-voting systems are not coercion-resistant, such as snapshot [29], Helios [2], and prêt à voter [28]. What's worse, in the blockchain context, vote-buying becomes easier with the help of smart contracts.

To address the coercion/vote-buying problem, several schemes are proposed. Generally, coercion-resistant voting can be divided into three categories: fake credentials [4, 14, 24], re-vote [1, 21, 25, 26], and secure hardware [3, 27]. In a coercion-resistant voting scheme using *fake credentials*, a voter holds both real and fake credentials. If coerced, a voter will cast a ballot using a fake credential, which is indistinguishable from the real one in the coercer's view, and it will be silently uncounted in the tally phase. In a *re-voting* scheme, a voter can cast his ballot multiple times and only the last one will be tallied. Coercion-resistance relies on that the voter can cast the ballot again after the coercer leaves. In the schemes using *secure hardware*, the secure hardware has its internal randomness source and can do probabilistic encryption for the voter so that the voter can lie about what has been encrypted.

Coercion-resistance v.s. deniability. All types of coercion-resistant voting schemes must give a voter deniability through some technique. Concretely, in "*fake credentials*" schemes, the election authority will provide randomness in the credential-related elements, and generate a designated verifier proof of correctness. To deceive the coercer, a voter can generate a fake credential and claim it as the real one by simulating the designated verifier proof. Namely, the registration procedure is deniable. In *re-vote* schemes, the re-vote operation must be deniable, i.e., the tally procedure will not reveal if a voter has re-voted. In the schemes using *secure hardware*, the secure hardware hides the randomness in the ciphertext so that a voter can claim that it is encryption of another candidate, i.e., the encryption operation is deniable.

Challenges. Could we apply the aforementioned techniques to realize a coercion-resistant voting scheme in the blockchain context? It turns out to be a non-trivial task. First of all, secure hardware based solutions might not be suitable for the blockchain setting, because an open blockchain allows anyone to join and leave freely and not all devices are equipped with a secure hardware, such as a trusted execution environment (TEE).

How about "fake credentials" and "re-voting" schemes? Can we adapt those schemes with differential voting power? There are still some challenges. For instance, JCJ [24] is a well-known coercion-resistance voting scheme, and it can be modified to support differential voting power; however, the scheme has $O(n^2)$ complexity due to the pair-wise plaintext equivalence tests (PETs), where n is the total number of votes, limiting its scalability. On the other hand, although the recently proposed scheme, VoteAgain [26], offers quasi-linear complexity, its verifiability relies on a trusted third party (TTP), which is undesirable in the blockchain setting. The best-known candidate is Araújo *et al.*'s "fake credentials" scheme [4], which achieves $O(n)$ complexity without relying on TTP for verifiability. Unfortunately, the credential of Araújo *et al.*'s scheme is in the form of two group elements satisfying a linear relationship, and this makes it unable to be modified trivially to support differential voting power. To the best of our knowledge, no proper coercion-resistant voting scheme in the literature can support differential voting power and achieve $O(n)$ complexity at the same time. Hereby, we are asking the question:

Can we design a scalable (linear complexity) coercion-resistant delegated voting scheme for blockchain decision-making?

1.1 Our Approach

In this work, we answer the above question affirmatively by proposing a new coercion-resistant voting scheme. Our scheme belongs to the "fake credentials" category. We start with the well-known JCJ scheme [24]. In the JCJ scheme, each encrypted credential is put on the bulletin board in the registration phase, and each ballot generally consists of an encrypted candidate and an encrypted credential. In the tally phase, by a shuffle and pair-wise PETs on the credentials, the ballots with fake credentials will be silently eliminated.

It is intuitive that one can associate credentials with voting power in the JCJ scheme to support differential voting power, i.e., each encrypted credential is tied with an encrypted voting power and we still perform PETs on the encrypted credentials in the tally phase. However, the scheme will have $O(n^2)$ complexity, so it does not scale well when the number of voters is large. To improve scalability, we propose a novel *"dummy voting power"* technique. The key idea is that we allow voters to publish (encrypted) fake credentials associated with (encrypted) zero voting power on the bulletin board. Then, in the tally phase, after shuffle re-encrypting the real and fake credentials, all the credentials can be decrypted. In this way, we transform the pair-wise PETs into "decrypt and matching", achieving $O(n)$ complexity (counting cryptographic operations only).

To achieve delegation, we design a *"two-layer homomorphic tally"* procedure consisting of "delegation calculation" and "final tally calculation". In layer one, delegation is calculated by decrypting voters' choices and adding the delegated voting power to the corresponding experts. In layer two, the final tally result is calculated by decrypting experts' choices and adding experts' voting power together with voters' direct votes. Thanks to the additive homomorphism of the encryption scheme, voters' ballots and voting power are hidden throughout the tally.

Combining the "dummy voting power" technique and "two-layer tally" procedure together, we build the first coercion-resistant voting scheme that has linear complexity and supports private differential voting power and liquid democracy. We perform the security analysis under the UC (universal

Table 1: Comparison of voting schemes. Here, n is the number of voters and m is the number of election candidates. Del. means delegation. Crypto state means that the voter needs to keep cryptographic secret from the coercer. EA stands for election authority. Hardware means that the property is guaranteed by secure hardware. An item in bold text means that our scheme is the best in this aspect.

	Diff. voting power	Del.	Ballot size	Complexity	Crypto state	Ballot privacy	Verifiability	Coercion-resistant
JCJ [24]	No	No	$O(1)$	$O(n^2)$	Yes	t-out-of-k	trust no one	trust EA
ABBT [4]	No	No	$O(1)$	$O(n)$	Yes	t-out-of-k	trust no one	trust EA
AKL+ [1], LHK [25]	No	No	$O(1)$	$O(n^2)$	Yes	t-out-of-k	trust no one	secret credential
VoteAgain [26]	No	No	$O(1)$	$O(n \log n)$	No	t-out-of-k	trust EA	trust EA
Secure hardware [3, 27]	No	No	$O(1)$	$O(n)$	No	hardware	hardware	hardware
Snapshot [29]	Yes	No	$O(1)$	$O(n)$	-	t-out-of-k	trust no one	-
ZOB [31]	Yes	Yes	$O(m)$	$O(mn)$	-	t-out-of-k	trust no one	-
Our scheme	**Yes**	**Yes**	$\boldsymbol{O(1)}$	$\boldsymbol{O(n)}$	Yes	**t-out-of-k**	**trust no one**	trust EA

composable) framework, and we prove that our scheme is UC coercion-resistant [3]. We implement the scheme and evaluate its performance. Results show that our scheme's tally execution time is more than 6x faster than VoteAgain [26] in elections with over 10,000 voters and over 50% extra ballot rate[1].

Note: *Vote-buying via stake-buying.* In blockchain voting, typically a voter's voting power is proportional to his stake. Since blockchain coins are publicly traded in open exchanges, one may argue that it is always possible to realize vote-buying via stake-buying. However, acquiring sufficient voting power through stake-buying is impractical. For an adversary to be successful, he needs to purchase a substantial amount of stake. Here's the catch: When buying from an exchange, there is often limited availability of stakes. Furthermore, rapidly purchasing large volumes of stake will inevitably drive up the price due to the basic principles of supply and demand. This surge in price could put the adversary's capital at risk. In contrast, vote-buying is a much simpler method and is detrimental to the decision-making process. Our coercion-resistant voting scheme prevents vote-buying on the blockchain.

1.2 Related Work

Coercion-resistant voting can be roughly split into three classes: fake credentials [4, 8, 11, 14, 15, 24], re-vote [1, 21, 25, 26], and secure hardware [3, 27]. JCJ [24] is the first paper that introduces the "fake credentials" type of coercion-resistant voting. In JCJ, each ballot contains an encrypted credential and there is a list of encrypted valid credentials in the bulletin board. In the tally phase, by pair-wise plaintext equivalence tests (PETs), the ballots with invalid credentials will be eliminated, but the pair-wise PETs obviously have $O(n^2)$ complexity. Other "fake credentials" schemes such as [4, 14] improve the time complexity and achieve better properties such as everlasting privacy. The re-voting type of coercion-resistant voting allows a voter to cast multiple ballots and the tally procedure will only count the last one. Achenbach *et al.* [1] and Locher *et al.* [25] utilize a deniable vote update mechanism to realize re-voting with quadratic complexity. The Norwegian Internet voting protocol [21] and VoteAgain [26] achieve (quasi-)linear complexity, but they both need a trusted third party for verifiability. Schemes based on secure hardware [3, 27] are easy to achieve coercion-resistance, but secure hardware is a strong assumption.

On the other hand, blockchain voting is becoming more and more popular nowadays. Snapshot [29] is a popular DAO (Decentralized Autonomous Organization) voting platform that frees voters from gas fees. It uses IPFS [7] to store the proposals and votes, making the voting process off-chain and gas-free. Zhang *et al.* [31] proposes a treasury system for blockchain governance. It supports liquid democracy and is provably secure, but its ballot size is linear to the candidate number. Besides, to the best of our knowledge, none of the existing blockchain voting schemes are coercion-resistant.

Finally, Table 1 gives a comparison between our scheme and previous work.

2 Preliminaries

Notations. Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\mathbb{G}$ be a cyclic group of prime order p with group generator g. We abbreviate *probabilistic polynomial time* as PPT.

2.1 (Lifted) ElGamal Encryption

ElGamal encryption scheme consists of three PPT algorithms: the key generation algorithm $\mathsf{EC.Keygen}(\mathbb{G}, g, p)$ takes as input the group parameters and outputs a public-private key pair $(\mathsf{pk} := g^{\mathsf{sk}}, \mathsf{sk})$; the encryption algorithm

[1]In VoteAgain, it means that more than 50% voters re-voted once; in our scheme, it means that more than 50% voters cast a fake ballot.

$\mathsf{EC.Enc}_{\mathsf{pk}}(m)$ takes as input the public key pk and the message $m \in \mathbb{G}$ and outputs the ciphertext $c := (c_1, c_2) := (g^r, m \cdot \mathsf{pk}^r)$; the decryption algorithm $\mathsf{EC.Dec}_{\mathsf{sk}}(c)$ takes as input the secret key sk and the ciphertext c and outputs the message $m := c_2 / c_1^{\mathsf{sk}}$.

ElGamal encryption is a re-randomizable encryption scheme. The re-encryption algorithm $\mathsf{EC.Rand}_{\mathsf{pk}}(c)$ takes as input a ciphertext $c := (c_1, c_2)$ and outputs the re-randomized ciphertext $c' := (c'_1, c'_2) := (g^r \cdot c_1, h^r \cdot c_2)$.

Lifted ElGamal encryption is a variant of ElGamal encryption. The encryption algorithm $\mathsf{LE.Enc}_{\mathsf{pk}}(m)$ takes as input the public key pk and the message m and outputs the ciphertext $c := (c_1, c_2) := (g^r, g^m \cdot \mathsf{pk}^r)$; the decryption algorithm $\mathsf{LE.Dec}_{\mathsf{sk}}(c)$ takes as input the secret key sk and the ciphertext c and outputs the message $m := \mathsf{Dlog}(c_2 / c_1^{\mathsf{sk}})$, where $\mathsf{Dlog}(x)$ outputs the discrete logarithm of x (note that computing the discrete logarithm is inefficient, thus the message space should be small in practice).

Clearly, the (lifted) ElGamal encryption scheme is IND-CPA secure under the DDH assumption (see Appendix B for formal definition). Lifted ElGamal encryption is additive homomorphic, i.e., $\mathsf{LE.Enc}_{\mathsf{pk}}(m_1) \cdot \mathsf{LE.Enc}_{\mathsf{pk}}(m_2) = \mathsf{LE.Enc}_{\mathsf{pk}}(m_1 + m_2)$. Besides, (lifted) ElGamal encryption can be distributed as a threshold encryption scheme [20].

2.2 Signature

A signature scheme Sig is defined by three PPT algorithms: A key generation algorithm $\mathsf{Sig.Keygen}(1^\lambda)$ that generates a public-private key pair $(\mathsf{pk}, \mathsf{sk})$; a signing algorithm $\sigma \leftarrow \mathsf{Sig.Sign}_{\mathsf{sk}}(m)$ that generates a signature on message m; and a verification algorithm $\mathsf{Sig.Verify}_{\mathsf{pk}}(\sigma, m)$ that outputs 1 if and only if σ is a valid signature on m.

A secure signature scheme is existentially unforgeable under chosen message attack (see Appendix B for formal definition).

2.3 Non-interactive Zero-knowledge Proof (NIZK)

A non-interactive zero-knowledge proof (NIZK) consists of four PPT algorithms: $\{\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim}\}$ and is complete, sound, and zero-knowledge (see Appendix B for formal definition). Our scheme utilizes six zero-knowledge proofs for proving: (i) voting power correctness ($\mathsf{NIZK}_{\mathsf{power}}$); (ii) ElGamal encryption plaintext knowledge ($\mathsf{NIZK}_{\mathsf{knowledge}}$); (iii) re-encryption correctness ($\mathsf{NIZK}_{\mathsf{DVF\text{-}reenc}}$); (iv) knowledge of secret key ($\mathsf{NIZK}_{\mathsf{sk}}$); (v) shuffle correctness ($\mathsf{NIZK}_{\mathsf{shuffle}}$); and (vi) decryption correctness ($\mathsf{NIZK}_{\mathsf{Dec}}$). We will give the details of these NIZKs in Appendix A.

Functionality $\mathcal{F}_{\mathsf{DKG}}^{t,k}[\mathbb{G}]$

The ideal functionality $\mathcal{F}_{\mathsf{DKG}}^{t,n}[\mathbb{G}]$ interacts with key generators $\mathcal{P} := \{P_1, \ldots, P_k\}$, an ideal adversary $\mathcal{S}$. It's parameterized with threshold t. Denote $\mathcal{P}_c$ as the set of corrupted generators, $\mathcal{P}_h := \mathcal{P} \setminus \mathcal{P}_c$ as the set of honest generators, and $|\mathcal{P}_c| \le t-1$. $\mathcal{F}_{\mathsf{DKG}}$ maintains a set $\mathcal{N}$(initially set to $\emptyset$).

Upon receiving $(\textsc{CorruptShares}, \mathsf{sid}, (\{j, \mathsf{sk}_j\}_{P_j \in P_c}))$ from $\mathcal{S}$:

- Pick $\mathsf{sk} \leftarrow \mathbb{Z}_q$, and compute $\mathsf{pk} := g^{\mathsf{sk}}$;
- Construct random polynomial $F(x) := \sum_{b=0}^{t-1} a_b \cdot x^b$ under the restriction $F(j) = \mathsf{sk}_j$ for $P_j \in \mathcal{P}_c$, and $F(0) = \mathsf{sk}$;
- Compute $\mathsf{sk}_i := F(i)$ and $\mathsf{ppk}_i = g^{\mathsf{sk}_i}$ for $i \in [n]$;
- Send $(\textsc{KeyGen}, \mathsf{sid}, \{\mathsf{pk}_j\}_{j \in [n]})$ to $\mathcal{S}$ and send $(\textsc{KeyGen}, \mathsf{sid}, \mathsf{sk}_i)$ to $P_i, i \in [n]$.

Upon receiving $(\textsc{ReadPk}, \mathsf{sid})$ from any party, return $(\textsc{ReadPk}, \mathsf{sid}, \mathsf{pk}, \{\mathsf{ppk}_i\}_{i \in [n]})$ to the requestor.

Figure 1: DKG ideal functionality $\mathcal{F}_{\mathsf{DKG}}^{t,k}[\mathbb{G}]$

Functionality $\mathcal{G}_{\mathsf{PBB}}$

The ideal functionality $\mathcal{G}_{\mathsf{PBB}}$ is globally available to all participants. It is parameterized with a predicate $\mathsf{Validate}$.

Upon initialization, set $\mathsf{Storage} := \emptyset$.
Upon receiving $(\textsc{Read}, \mathsf{sid})$ from P:

- let $\mathsf{val} := \mathsf{Storage}[\mathsf{sid}]$;
- return $(\textsc{Read}, \mathsf{sid}, \mathsf{val})$ to the requestor.

Upon receiving $(\textsc{Write}, \mathsf{sid}, \mathsf{inp})$ from P, do the following:

- let $\mathsf{val} := \mathsf{Storage}[\mathsf{sid}]$;
- if $\mathsf{Validate}(\mathsf{val}, \mathsf{inp}) = 1$, then set $\mathsf{Storage}[\mathsf{sid}] := \mathsf{val} || \mathsf{inp}$, return $(\textsc{Receipt}, \mathsf{sid})$ to the requestor;
- Otherwise, return $(\textsc{Reject}, \mathsf{sid})$ to the requestor.

Figure 2: Functionality $\mathcal{G}_{\mathsf{PBB}}$

Functionality $\mathcal{F}_{\mathsf{sc}}$

The ideal functionality $\mathcal{F}_{\mathsf{sc}}$ proceeds as follows.

- Upon receiving $(\textsc{Send}, \mathsf{sid}, R, m)$ from a party S, send $(\textsc{Sent}, \mathsf{sid}, S, R, |m|)$ to the adversary $\mathcal{S}$ and send $(\textsc{Sent}, \mathsf{sid}, S, m)$ to R.
- Ignore other requests.

Figure 3: Functionality $\mathcal{F}_{\mathsf{sc}}$

2.4 Universal Composibility

We perform security analysis under the Universally Composable (UC) framework [9, 10]. In the UC framework, a protocol is represented by a set of interactive Turing machines (ITMs). Each ITM contains the program to be run by a party. Security is based on the indistinguishability between the real world execution $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ and the ideal world execution $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$. In $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$, the parties run protocol Π with the adversary $\mathcal{A}$. In $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, the parties interact with the ideal functionality $\mathcal{F}$ with the ideal adversary (simulator) $\mathcal{S}$. If for all PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that no PPT environment $\mathcal{Z}$ can distinguish the real world and the ideal world, then we say that the protocol Π UC-realizes the ideal functionality $\mathcal{F}$.

2.5 Distributed Key Generation

Our voting scheme utilizes a distributed key generation protocol for threshold key generation. We use an ideal functionality $\mathcal{F}_{\mathsf{DKG}}^{t,k}$ [30] to abstract the DKG procedure. The functionality $\mathcal{F}_{\mathsf{DKG}}^{t,k}$ is depicted in Fig. 1. It interacts with key generators $\mathcal{P} := \{P_1, \dots, P_k\}$ to generate a public key pk and deal the secret key shares sk_j to P_j. Meanwhile, it publishes each party P_i's partial public key ppk_i. To realize $\mathcal{F}_{\mathsf{DKG}}^{t,k}$, we can use the threshold distributed key generation protocol proposed by Gennaro *et al.* [20].

2.6 Public Bulletin Board

For a voting scheme, a public bulletin board is needed for broadcasting the ballots and other auxiliary information such as zk proofs. Formally, we use a shared functionality [10] $\mathcal{G}_{\mathsf{PBB}}$ to model the public bulletin board, as depicted in Fig. 2. $\mathcal{G}_{\mathsf{PBB}}$ has two interfaces: READ and WRITE, and it guarantees the anonymity of the sender. In practice, the blockchain serves as the public bulletin board.

2.7 Secure Channel

Coercion-resistant voting requires a secure channel between a voter and the authority. We model it as a UC secure channel functionality $\mathcal{F}_{\mathsf{sc}}$, as depicted in Fig. 3. Note that $\mathcal{F}_{\mathsf{sc}}$ only deals with transmission of a single message. Secure channel for multiple messages is obtained by invoking multiple sessions of $\mathcal{F}_{\mathsf{sc}}$.

3 System Overview

In this section, we start by modifying the JCJ protocol [24] to build a coercion-resistant voting scheme that supports differential voting power with $O(n^2)$ complexity. Then, we give the intuition and details of our novel "dummy voting power" technique and "two-layer tally" procedure. Finally, we provide an overview of our scheme.

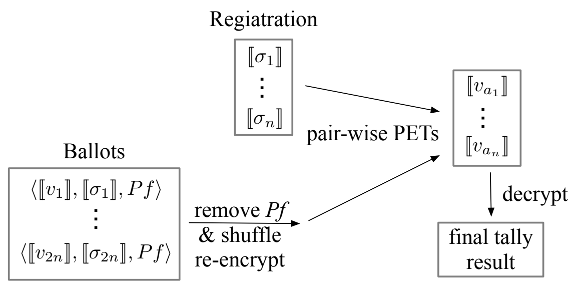
Regiatration
$[\![\sigma_1]\!]$ ⋮ $[\![\sigma_n]\!]$
pair-wise PETs
$[\![v_{a_1}]\!]$ ⋮ $[\![v_{a_n}]\!]$
Ballots
$\langle[\![v_1]\!],[\![\sigma_1]\!],Pf\rangle$ ⋮ $\langle[\![v_{2n}]\!],[\![\sigma_{2n}]\!],Pf\rangle$
remove Pf & shuffle re-encrypt
decrypt
final tally result

Figure 4: JCJ original protocol

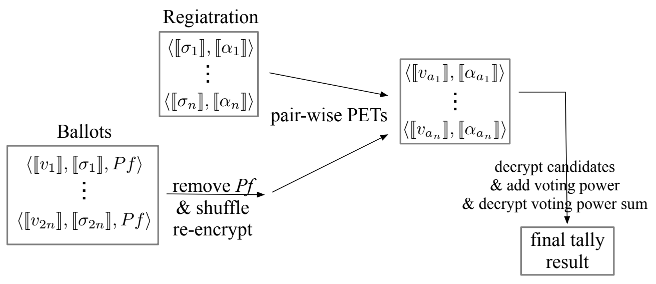
Regiatration
$\langle[\![\sigma_1]\!],[\![\alpha_1]\!]\rangle$ ⋮ $\langle[\![\sigma_n]\!],[\![\alpha_n]\!]\rangle$
pair-wise PETs
$\langle[\![v_{a_1}]\!],[\![\alpha_{a_1}]\!]\rangle$ ⋮ $\langle[\![v_{a_n}]\!],[\![\alpha_{a_n}]\!]\rangle$
Ballots
$\langle[\![v_1]\!],[\![\sigma_1]\!],Pf\rangle$ ⋮ $\langle[\![v_{2n}]\!],[\![\sigma_{2n}]\!],Pf\rangle$
remove Pf & shuffle re-encrypt
decrypt candidates & add voting power & decrypt voting power sum
final tally result

Figure 5: JCJ protocol with differential voting power

3.1 The JCJ Protocol

We first recall the well-known JCJ protocol [24] and show how to modify it to support differential voting power.

The original protocol. As mentioned above, JCJ is the first protocol that introduces the concept of "fake credentials". Generally, it works as follows. For simplicity, we use $[\![x]\!]$ to denote encryption of x in this sec. 3.1 and 3.2. In the registration phase, a voter authenticates to the election authority (EA) and the EA generates a credential $\sigma \leftarrow \mathbb{G}$. Then, the EA publishes $S = [\![\sigma]\!]$ on the PBB and sends σ to the voter along with a designated verifier proof that S is encryption of σ. In the voting phase, a voter cast a ballot $B = ([\![v]\!], [\![\sigma]\!], Pf)$ where $[\![v]\!]$ is the encryption of a candidate v, Pf includes the NIZK proofs of knowledge of v and σ, and a NIZK proof that $[\![v]\!]$ encrypts a valid candidate. If a voter is coerced, he generates a random $\sigma' \leftarrow \mathbb{G}$ and claims it as the real credential by simulating the designated verifier proof. In the tally phase, the trustees shuffle the ballots and perform pair-wise PETs on encrypted credentials to eliminate the ballots with fake credentials. Finally, the trustees decrypt the candidates and tally the votes. The overview of JCJ protocol is shown in Fig. 4.

Supporting differential voting power. We can see that if we associate each credential with voting power, then the system can easily support differential voting power. More specifically, in the registration phase, the EA publishes $S = ([\![\sigma]\!], [\![\alpha]\!])$ on the PBB, where $[\![\sigma]\!]$ is the encrypted credential and $[\![\alpha]\!]$ is the encrypted voting power under an additive homomorphic encryption scheme. After the pair-wise PETs, we get tuples of $\langle [\![v]\!], [\![\alpha]\!] \rangle$, where $[\![v]\!]$ is the encrypted candidate and $[\![\alpha]\!]$ is the encrypted voting power. Then, the trustees decrypt the candidates and add the voting power by additive homomorphism. The overview of the modified JCJ protocol with differential voting power is shown in Fig. 5.

3.2 Our Technique

In this part, we will illustrate our novel techniques that can achieve $O(n)$ complexity and delegated voting.

Dummy voting power. We can see that in the JCJ protocol, the pair-wise PETs cause $O(n^2)$ complexity. The idea is that: if we allow voters to publish the fake credentials on the PBB, but associate them with dummy (zero) voting power, then we can directly decrypt the credentials instead of performing PETs in the tally phase. In other words, in the registration phase, the EA publishes (encrypted) real credentials and (encrypted) real voting power; at any convenient time, voters can also publish (encrypted) fake credentials and (encrypted) dummy voting power. In this way, we switch pair-wise PET into shuffle-decrypt and matching, achieving linear complexity. Furthermore, "dummy voting power" also hides the number of votes obtained by each candidate. The idea of "dummy voting power" technique is shown in Fig. 6.

Two-layer tally. To support delegation, the trustees perform a "two-layer tally" in the tally phase. Generally speaking, in layer one, voters' choices are decrypted and the delegated voting power will be added to the corresponding experts; in layer two, experts' choices are decrypted and the final tally result is calculated by adding experts' voting power and voters' direct votes. Note that, experts have input independence instead of ballot privacy so their ballots can be decrypted directly. Fig. 7 shows the process of a "two-layer tally" procedure where there are 3 candidates and 2 experts.

3.3 Overview of Our Scheme

In this section, we define the roles in our system and provide an overview of our scheme in the blockchain context.

Roles. There are five roles in the protocol: voters, experts, registration authority (RA), shuffler, and trustees.

- *A voter* has a certain amount of voting power and can either vote on the proposal directly or delegate his voting power to an expert.
- *An expert* does not have voting power himself, but he can be delegated to vote on others' behalf.
- *The RA* is responsible for the registration procedure.

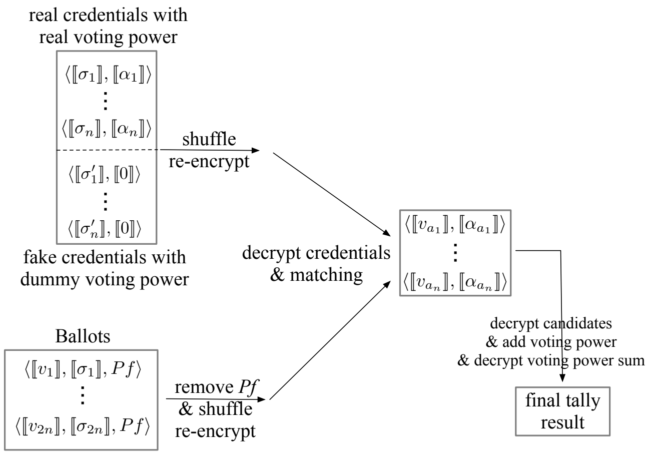
real credentials with real voting power
$\langle [\![\sigma_1]\!], [\![\alpha_1]\!] \rangle$
$\langle [\![\sigma_n]\!], [\![\alpha_n]\!] \rangle$
$\langle [\![\sigma'_1]\!], [\![0]\!] \rangle$
$\langle [\![\sigma'_n]\!], [\![0]\!] \rangle$
fake credentials with dummy voting power
shuffle
re-encrypt
decrypt credentials & matching
$\langle [\![v_{a_1}]\!], [\![\alpha_{a_1}]\!] \rangle$
$\langle [\![v_{a_n}]\!], [\![\alpha_{a_n}]\!] \rangle$
decrypt candidates & add voting power & decrypt voting power sum
final tally result
Ballots
$\langle [\![v_1]\!], [\![\sigma_1]\!], Pf \rangle$
$\langle [\![v_{2n}]\!], [\![\sigma_{2n}]\!], Pf \rangle$
remove Pf & shuffle re-encrypt

Figure 6: Dummy voting power technique

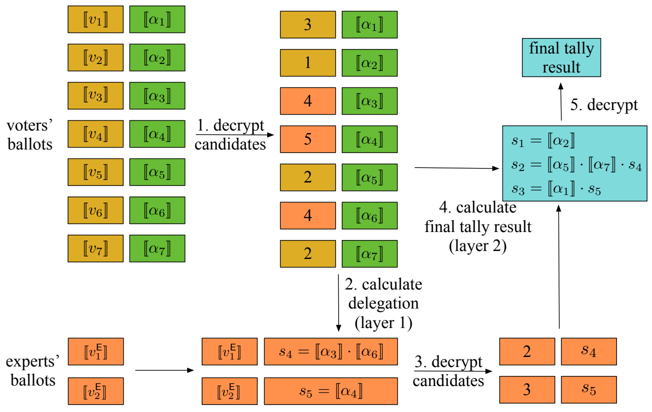
voters' ballots
$[\![v_1]\!]$ $[\![\alpha_1]\!]$
$[\![v_2]\!]$ $[\![\alpha_2]\!]$
$[\![v_3]\!]$ $[\![\alpha_3]\!]$
$[\![v_4]\!]$ $[\![\alpha_4]\!]$
$[\![v_5]\!]$ $[\![\alpha_5]\!]$
$[\![v_6]\!]$ $[\![\alpha_6]\!]$
$[\![v_7]\!]$ $[\![\alpha_7]\!]$
1. decrypt candidates
3 $[\![\alpha_1]\!]$
1 $[\![\alpha_2]\!]$
4 $[\![\alpha_3]\!]$
5 $[\![\alpha_4]\!]$
2 $[\![\alpha_5]\!]$
4 $[\![\alpha_6]\!]$
2 $[\![\alpha_7]\!]$
2. calculate delegation (layer 1)
experts' ballots
$[\![v^E_1]\!]$
$[\![v^E_2]\!]$
$[\![v^E_1]\!]$ $s_4 = [\![\alpha_3]\!] \cdot [\![\alpha_6]\!]$
$[\![v^E_2]\!]$ $s_5 = [\![\alpha_4]\!]$
3. decrypt candidates
2 s_4
3 s_5
4. calculate final tally result (layer 2)
$s_1 = [\![\alpha_2]\!]$
$s_2 = [\![\alpha_5]\!] \cdot [\![\alpha_7]\!] \cdot s_4$
$s_3 = [\![\alpha_1]\!] \cdot s_5$
5. decrypt
final tally result

Figure 7: Two-layer tally

- *The shuffler* verifiably shuffles the ballots and public keys.
- *The trustees* are responsible for decrypting the ballot and revealing the final tally result.

We use n, m, ℓ to denote the voter number, expert number, and candidate number, respectively. There are k trustees with threshold t.

An optimization of JCJ's ballot structure. We optimize JCJ's ballot structure in the following two aspects. Firstly, we observe that it is not necessary to prove that $[\![v]\!]$ encrypts a valid candidate because all the selections will be decrypted in the tally phase. If it encrypts an invalid value, we can simply treat it as "abstain" and drop it. Secondly, a slight modification is performed to reduce the ballot size: instead of defining σ as the secret credential, we can define $\sigma := \mathsf{pk} := g^{\mathsf{sk}}$ and let the voter keep the secret key sk. Then, the ballot can be modified as $\langle \mathsf{pk}, \boldsymbol{u}, \pi, Sig \rangle$, where $\boldsymbol{u} := [\![v]\!]$ is the encrypted choice, $Sig \leftarrow \mathsf{Sign}(\mathsf{sk}, \boldsymbol{u})$, and π is a NIZK proof of plaintext knowledge of $\boldsymbol{u}$. After matching the choices with

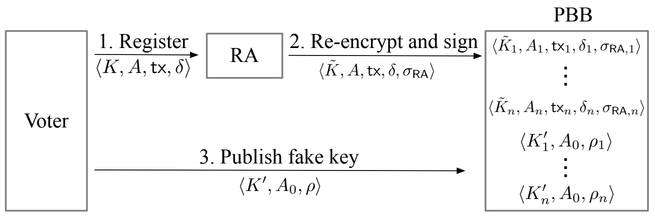
PBB
1. Register
$\langle K, A, \mathsf{tx}, \delta\rangle$
RA
2. Re-encrypt and sign
$\langle \tilde{K}, A, \mathsf{tx}, \delta, \sigma_{\mathsf{RA}}\rangle$
Voter
$\langle \tilde{K}_1, A_1, \mathsf{tx}_1, \delta_1, \sigma_{\mathsf{RA},1}\rangle$
$\langle \tilde{K}_n, A_n, \mathsf{tx}_n, \delta_n, \sigma_{\mathsf{RA},n}\rangle$
3. Publish fake key
$\langle K', A_0, \rho\rangle$
$\langle K'_1, A_0, \rho_1\rangle$
$\langle K'_n, A_0, \rho_n\rangle$

Figure 8: Registration phase

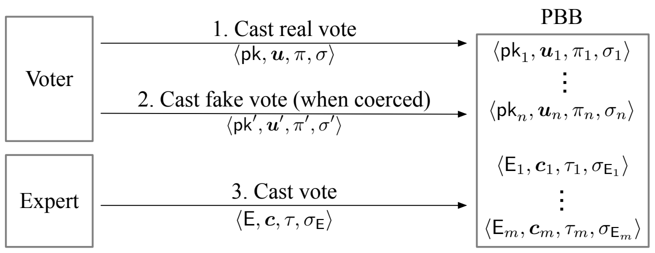
PBB
Voter
1. Cast real vote
$\langle \mathsf{pk}, \boldsymbol{u}, \pi, \sigma\rangle$
2. Cast fake vote (when coerced)
$\langle \mathsf{pk}', \boldsymbol{u}', \pi', \sigma'\rangle$
Expert
3. Cast vote
$\langle \mathsf{E}, \boldsymbol{c}, \tau, \sigma_{\mathsf{E}}\rangle$
$\langle \mathsf{pk}_1, \boldsymbol{u}_1, \pi_1, \sigma_1\rangle$
$\langle \mathsf{pk}_n, \boldsymbol{u}_n, \pi_n, \sigma_n\rangle$
$\langle \mathsf{E}_1, \boldsymbol{c}_1, \tau_1, \sigma_{\mathsf{E}_1}\rangle$
$\langle \mathsf{E}_m, \boldsymbol{c}_m, \tau_m, \sigma_{\mathsf{E}_m}\rangle$

Figure 9: Voting/delegation phase

voting power, the trustees shuffle and decrypt the choices (see Fig. 10 for the whole tally procedure). By doing so, we change an Elgamal encryption $[\![\sigma]\!]$ to a group element pk, achieving smaller ballot size.

Overview. Our voting scheme has four phases: preparation phase, registration phase, voting/delegation phase, and tally phase.

In the preparation phase, the RA generates a public-private signing key pair $\langle \mathsf{pk}_{\mathsf{RA}}, \mathsf{sk}_{\mathsf{RA}}\rangle$. The trustees perform a distributed key generation protocol to generate pk_{T} and share sk_{T}.

In the registration phase (see Fig. 8), each voter first freezes some stake by transaction tx. Then, he generates a pair of real voting keys (serving as the real credential) $\langle \mathsf{pk}, \mathsf{sk}\rangle$ and sends a registration message $\langle K, A, \mathsf{tx}, \delta\rangle$ to the RA (step 1), where $K \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_{\mathsf{T}}}(\mathsf{pk})$ is encryption of real public key, $A \leftarrow \mathsf{LE.Enc}_{\mathsf{pk}_{\mathsf{T}}}(\alpha)$ is encryption of voting power, tx is the transaction that freezes some stake, and δ is a NIZK proof that the encrypted voting power equals the frozen stake (Cf. Appendix. A). After authenticating the voter (i.e., checking that the voter knows the sk corresponding to tx's sender's pk by an interactive zero-knowledge protocol), the RA re-encrypts K as $\tilde{K}$ and signs the registration message. Then, the RA sends a designated verifier proof of re-encryption correctness to the voter, and sends $\langle \tilde{K}, A, \mathsf{tx}, \delta, \sigma_{\mathsf{RA}}\rangle$ to the PBB (step 2), where $\sigma_{\mathsf{RA}} \leftarrow \mathsf{Sig.sign}_{\mathsf{sk}_{\mathsf{RA}}}(\tilde{K}||A||\mathsf{tx}||\delta)$. At any convenient time, the voter can generate a pair of fake voting keys (serving as the fake credential) $\langle \mathsf{pk}', \mathsf{sk}'\rangle$ and publish a fake key item $\langle K', A_0, \rho\rangle$ on the PBB (step 3), where $K' \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_{\mathsf{T}}}(\mathsf{pk}')$ is encryption of fake public key, A_0 is a deterministic encryption of 0, and ρ is a NIZK proof of knowledge of the corresponding secret key. The voter can repeat step 3 multiple times to generate multiple fake keys.

In the voting phase (see Fig. 9), each voter encrypts his choice with the trustees' public key pk_{T}, signs it with the voting secret key and casts it on PBB (step 1). Specifically, a voter's ballot is of the form $\langle \mathsf{pk}, \boldsymbol{u}, \pi, \sigma\rangle$, where $\boldsymbol{u} \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_{\mathsf{T}}}(v)$ is the encrypted choice, $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \boldsymbol{u})$ is the signature and π is a NIZK proof of plaintext knowledge of $\boldsymbol{u}$. If a voter is coerced, he will use the fake key pair $\langle \mathsf{pk}', \mathsf{sk}'\rangle$ to perform the voting process (step 2). Thanks to the re-encryption by RA and the designated verifier proof, the voter can claim that $\tilde{K}$ is re-encryption of $\mathsf{EC.Enc}_{\mathsf{pk}_{\mathsf{T}}}(\mathsf{pk}')$ by simulating the designated verifier proof. In this phase, each expert also casts his ballot by simply encrypting the choice and signing it with his blockchain secret key (step 3), i.e., an expert's ballot is of the form $\langle \mathsf{E}, \boldsymbol{c}, \tau, \sigma_{\mathsf{E}}\rangle$, where E is the expert's identity, $\boldsymbol{c} \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_{\mathsf{T}}}(v^{\mathsf{E}})$ is the encrypted candidate, σ_{E} is the signature and τ is the NIZK proof of plaintext knowledge.

In the tally phase (see Fig. 10), the PBB contains "encrypted public key items", voters' ballots and experts' ballots at the beginning. For simplicity, in the figure, we assume there are n real ballots and n fake ballots; in reality, a voter can cast any number of fake ballots. Firstly, the shuffler checks the validity of all the "encrypted public key items" and ballots, and removes the NIZK proofs and signatures (step 1). Then, the shuffler shuffle re-encrypts the "encrypted public key items" (step 2). Next, the trustees jointly decrypt the public keys in "encrypted public key items" and voters' ballots (step 3). If the same public key appears more than once, then drop them. The encrypted voting power and encrypted choices with the same public key will be matched (step 4). To ensure ballot privacy, the matched items need to be shuffle re-encrypted (Step 5). Next, the trustees jointly decrypt voters' choices (step 6) and experts' choices (step 7). After the decryption, the trustees will add the voting power to the corresponding candidates by a two-layer tally (step 8). In layer 1, experts' obtained voting power is calculated, i.e., expert E_i's obtained voting power is $\boldsymbol{s}_i := \prod_{v_j=\ell+i} A_{c_j}, i \in [1,m]$, where ℓ is the candidate number and m is the expert number; in layer 2, the votes for each candidate are tallied by adding direct votes and expert votes together, i.e., candidate v's obtained voting power is $\boldsymbol{s}_v := \prod_{v_j=v} A_{c_j} \prod_{v_j^{\mathsf{E}}=v} \boldsymbol{s}_j, v \in [1,l]$. Ballots that encrypt an invalid choice will be dropped. Finally, the trustees jointly decrypt $\{\boldsymbol{s}_v\}_{v\in[1,\ell]}$ to publish the final tally result (step 9).

Blockchain deployment. To deploy the scheme on a blockchain, we need to select the RA, shuffler, and trustees in the blockchain context. Also, there should be a validator that checks all the NIZK proofs.

The RA. Note that the communication between the voter and the RA must be secret to the coercer. Also, the RA is trusted for coercion-resistance and cannot be distributed (Cf. sec. 4.1). Therefore, it may be instantiated with trusted execution environment (TEE) like Intel SGX.

The shuffler. The shuffler can be implemented by a

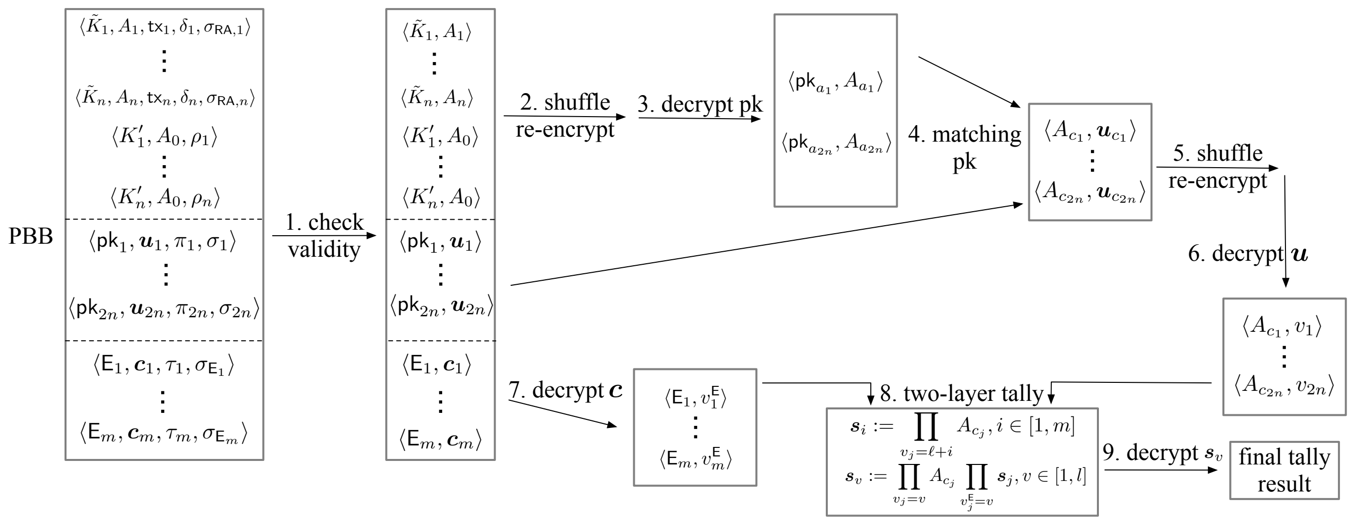

Figure 10: Tally phase

mixnet [12], and the mixnet nodes can be selected by cryptographic sortition [13]. The shuffle is secure as long as one mixnet node is honset.

The trustees. The trustees can also be selected by cryptographic sortition [13]. The majority of trustees are honest with a high probability when a majority of the stake is honest.

The validator. Every participant can be the validator to check all the NIZK proofs if he wants. Since there is shuffle proofs in our scheme, whose verification cost is relatively heavy, it is not recommended to deploy a smart contract to play the role of the validator.

Roles of the blockchain. In our scheme, the blockchain serves as the PBB. It also plays the role of PKI in the registration phase to authenticate a voter.

4 Assumptions and Security Modeling

In this section, we first informally define ballot privacy, verifiability, and coercion-resistant and we analyze under which assumptions these properties are achieved. Then, we formally introduce the UC incoercibility framework.

4.1 Security Properties and Assumptions

Definition 1. (Ballot privacy) The adversary cannot learn the votes of honest voters.

Definition 2. (Coercion-resistance) A coercer cannot determine if the coerced party is trying to deceive him.

Definition 3. (Verifiability) Honest voters' ballots must be tallied and the adversary cannot cast more votes than the number of voters that he controls.

Assumptions. Our scheme only relies on basic assumptions that any "fake credentials" type of coercion-resistant voting scheme needs. We list these assumptions, explain the necessity, and discuss how they are achieved in the blockchain context.

Assumption 1. The public bulletin board is honest and the communication with the PBB is anonymous.

PBB is a basic assumption for any electronic voting scheme. A malicious PBB can break verifiability by creating different views for different voters [22]. Then, since ballots can be dropped undetectably, ballot privacy will be undermined [16], and it is not possible to have coercion-resistance without ballot privacy. Thus, PBB is trusted for all three properties. Moreover, communication with PBB must be anonymous; otherwise, the coercer will catch the deceiving voter when he tries to cast the real ballot.

In our system, the blockchain is a public ledger and serves as the honest PBB. A voter can use anonymous channels (e.g., TOR) to broadcast on the blockchain.

Assumption 2. There is a secure (untappable) channel between the voter and the RA.

In all "fake credentials" schemes, a voter needs to establish a secret in the registration phase and keep the secret from the coercer. If the coercer taps all the communication between the voter and the authorities, then the voter's private information is a receipt/witness of what he cast [23].

As mentioned above, the RA can be instantiated with TEE like Intel SGX. A voter can use TOR to communicate with the RA.

Assumption 3. The authentication is inalienable [1], i.e., the coercer cannot impersonate the voter or stop the voter from authenticating.

Inalienable authentication is a must for all voting schemes. Otherwise, the adversary can vote on the voter's behalf or launch a forced abstention attack.

In the blockchain context, the blockchain plays the role of PKI and each voter authenticates to the RA by proving knowledge of his blockchain secret key. It is assumed that a voter will not reveal his blockchain secret key to the coercer, which will put the voter's stake at risk.

In the following, we analyze how ballot privacy, verifiability, and coercion-resistance are achieved.

Ballot privacy. In the registration phase, the voter himself generates the voting public key, and he encrypts it with the trustees' public key before sending it to the RA. Thus, nobody knows the link between the voting public key and the voter as long as the majority of trustees are honest. In the voting/delegation phase, voters' ballots are also encrypted with the trustees' public key. In the tally phase, ballots and "encrypted public key items" will be shuffled before decryption so that the link between identity and voting public key is broken by the shuffle. Therefore, ballot privacy is achieved if the shuffler and the majority of trustees are honest.

Note: In delegated voting, usually the experts have input independence rather than ballot privacy, i.e., when casting the ballot, it should be independent of the others; later in the tally phase, it will be decrypted directly without shuffle. This is an important requirement for delegated voting because we want to detect if an expert's behavior deviates from what he claimed.

Verifiability. A process composed of several subroutines is verifiable if each subroutine is verifiable itself. In the preparation phase, the trustees perform a verifiable distributed key generation protocol [20]. In the registration phase, we use two NIZKs to ensure that (i) the encrypted voting power is equal to the frozen stake; (ii) the RA does the re-encryption correctly. In the voting/delegation phase, EUF-CMA property of the signature scheme prevents anyone who does not know the secret key from casting a valid ballot. In the tally phase, the shuffle correctness is guaranteed by the shuffle NIZK [5], and decryption correctness is guaranteed by the decryption NIZK [20]. In conclusion, all subroutines in our scheme are publicly verifiable so no one needs to be trusted for verifiability.

Coercion-resistance. A coercer may ask the voter to reveal his real voting key pair, but the voter can claim a fake key pair as real by simulating the designated verifier proof, as long as the RA is not colluding with the coercer. In the tally phase, after shuffling all the "encrypted public key items", real keys and fake keys become indistinguishable from the coercer's perspective. Besides, the majority of trustees must be honest to ensure that the coercer cannot decrypt the ciphertexts.

Finally, Table 2 summarizes the trust assumptions on the entities for achieving each property.

Notes on distributing the shuffler and RA: To distribute the shuffler, a mixnet [12] can be utilized and we can perform a cryptographic sortition [13] on the blockchain to select the mixnet nodes. The assumption becomes that at least one of the mixnet nodes is honest instead of trusting a single shuffler.

Table 2: Trust assumptions on the entities.

	Ballot privacy	Verifiability	Coercion-resistance
PBB	Trusted	Trusted	Trusted
RA	Untrusted	Untrusted	Trusted
Shuffler	Trusted	Untrusted	Trusted
Trustees	t-out-of-k	Untrusted	t-out-of-k

However, simply distributing the RA does not lead to a weaker assumption because the coercer can ask the voter to provide the entire view of the registration phase. Even if only one of the RA parties is colluding with the coercer, the voter who does not know which RA party is colluding cannot simulate the registration view with negligible fail probability. Concretely, if the voter fakes a message sent by a RA member, then he will have at least $1/n$ probability of being caught (in the case that the RA member is malicious), where n is the number of RA parties. Therefore, it is better not to distribute the RA for "fake credentials" schemes. We suggest using TEE to instantiate the RA on the blockchain.

4.2 Security Modeling

Framework. We formally perform security analysis under the Universal Composable (UC) framework [9]. As mentioned in sec. 2.4, security is based on the indistinguishability between the ideal world and the real/hybrid world.

Modeling coercion-resistance. We adopt the UC incoercibility definition proposed by Alwen, Ostrovsky, Zhou, and Zikas (AOZZ) [3]. In their definition, the ideal deception strategy DI in the ideal world is controlled by the environment $\mathcal{Z}$ and the ideal adversary $\mathcal{S}$ plays the role of coercer. When DI receives an input x from the ideal adversary (coercer) $\mathcal{S}$, the environment $\mathcal{Z}$ maps x to x' and instructs DI to forward x' to the ideal functionality $\mathcal{F}$ (x' can be equal to x). In the real world, the real deception strategy DR is a twist on the protocol Π and DR internally runs DI. DR will interact with the real adversary (coercer) $\mathcal{A}$ and do actions according to DI's output.

We can see that, in the ideal world, the ideal deceiving strategy can map the coercer's input x to any other input x', which represents "cast-as-intend". Also, the ideal adversary $\mathcal{S}$ determines if a party is deceiving no better than someone who only sees the outputs of the computation. Thus, if for every DI there exists DR that makes the ideal world and the real world indistinguishable, we say that the protocol Π is IUC (incoercible UC) secure.

The ideal world execution. In the ideal world, the voters $\mathcal{V}$, experts $\mathcal{E}$, trustees $\mathcal{T}$, shuffler, registration authority RA, and ideal adversary $\mathcal{S}$ communicate with the ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$ (Cf. Fig. 11). The ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$ has four phases: preparation phase, registration phase, voting/delegation phase, and tally phase.

Voting ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$

The functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$ interacts with a set of voters $\mathcal{V} := \{\mathsf{V}_1,\ldots,\mathsf{V}_n\}$, a set of experts $\mathcal{E} := \{\mathsf{E}_1,\ldots,\mathsf{E}_m\}$, a set of trustees $\mathcal{T} := \{\mathsf{T}_1,\ldots,\mathsf{T}_k\}$, the shuffler, the registration authority RA and the adversary $\mathcal{S}$. It is parameterized with candidate number ℓ and threshold t and it internally keeps variables $\mathcal{J},\eta,\mathsf{state},\mathsf{ballots}$. Denote $\mathcal{T}_{\mathsf{cor}}$ and $\mathcal{T}_{\mathsf{honest}}$ as the set of corrupted and honest trustees, respectively. Denote the candidates as $\mathcal{C}$.
Initially, $\mathcal{J} := \eta := \mathsf{ballots} := \emptyset$, $\mathsf{state} := 0$.

Preparation phase.
Upon receiving $(\textsc{Init},\mathsf{sid})$ from the trustee T_i, send a notification message $(\textsc{InitNotify},\mathsf{sid},\mathsf{T}_i)$ to $\mathcal{S}$.
Upon receiving $(\textsc{Init},\mathsf{sid})$ from the RA, send a notification message $(\textsc{InitNotify},\mathsf{sid},\mathsf{RA})$ to $\mathcal{S}$.

Registration phase.
When $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$ receives $(\textsc{Init},\mathsf{sid})$ from all trustees and RA, set $\mathsf{state} := 1$.
Upon receiving $(\textsc{Reg},\mathsf{sid},\alpha_j)$ from the voter V_j, $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$ does the following: (ignore the request if $\mathsf{state} \neq 1$)

- Set $\mathsf{power}[\mathsf{V}_j] := \alpha_j$.
- If $|\mathcal{T} \cap \mathcal{T}_{\mathsf{cor}}| \geq t$, send $(\textsc{LeakPower},\mathsf{sid},\langle \alpha_i,\mathsf{V}_i\rangle)$ to $\mathcal{S}$.
- Otherwise, send a notification message $(\textsc{RegNotify},\mathsf{sid},\mathsf{V}_j)$ to the adversary $\mathcal{S}$.

Voting/Delegation phase:
When $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$ receives $(\textsc{Reg},\mathsf{sid})$ from all voters, set $\mathsf{state} := 2$.
Upon receiving $(\textsc{Vote},\mathsf{sid},v_i)$ from the voter V_i or the expert E_i, $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$ does the following: (ignore the request if $\mathsf{state} \neq 2$)

- Set $\mathsf{ballots}[\mathsf{V}_i(\text{or } \mathsf{E}_i)] := v_i$.
- If the request is from a voter, send $(\textsc{VoteNotify},\mathsf{sid},\mathsf{VOTER})$ to $\mathcal{S}$. Otherwise, send $(\textsc{VoteNotify},\mathsf{sid},\mathsf{EXPERT})$ to $\mathcal{S}$.
- If $|\mathcal{T} \cap \mathcal{T}_{\mathsf{cor}}| \geq t$, send $(\textsc{LeakVote},\mathsf{sid},\langle v_i,\mathsf{V}_i(\text{or } \mathsf{E}_i)\rangle)$ to $\mathcal{S}$.

Tally phase.
Upon receiving $(\textsc{VoteEnd},\mathsf{sid})$ from the shuffler, send $(\textsc{VoteEnd},\mathsf{sid})$ to the adversary $\mathcal{S}$, and set $\mathsf{state} := 3$.
Upon receiving $(\textsc{Tally},\mathsf{sid})$ from the trustee T_i, $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$ does the following: (ignore the request if $\mathsf{state} \neq 3$)

- Set $\mathcal{J} := \mathcal{J} \cup \{\mathsf{T}_i\}$, if $|\mathcal{J} \cap \mathcal{T}_{\mathsf{honest}}| + |\mathcal{T}_{\mathsf{cor}}| \geq t$, compute $\eta \leftarrow \mathsf{TallyAlg}(\mathcal{V},\mathcal{E},\mathcal{C},\mathsf{ballots},\mathsf{power})$ (Cf. Fig. 12) and $\mathsf{bc}_{\mathsf{V}} \leftarrow \mathsf{CountAlg}(\mathcal{V},\mathcal{E},\mathcal{C},\mathsf{ballots})$ (Cf. Fig. 13). Denote experts' ballots as $\mathsf{ballots}_{\mathsf{E}}$. Send $(\textsc{LeakTally},\mathsf{sid},\eta,\mathsf{ballots}_{\mathsf{E}},\mathsf{bc}_{\mathsf{V}})$ to the adversary $\mathcal{S}$.
- If $|\mathcal{J} \cap \mathcal{T}_{\mathsf{honest}}| + |\mathcal{T}_{\mathsf{cor}}| \geq t$ and the shuffler is corrupted, send $(\textsc{LeakBallots},\mathsf{sid},\mathsf{ballots})$ to the adversary $\mathcal{S}$.
- Send $(\textsc{TallyNotify},\mathsf{sid},\mathsf{T}_i)$ to adversary $\mathcal{S}$.
- If $|\mathcal{J}| \geq t$, set $\eta \leftarrow \mathsf{TallyAlg}(\mathcal{V},\mathcal{E},\mathcal{C},\mathsf{ballots},\mathsf{power})$ (Cf. Fig. 12).

Upon receiving $(\textsc{ReadTally},\mathsf{sid})$ from any party, return $(\textsc{ReadTallyReturn},\mathsf{sid},\eta)$ to the requestor.

Figure 11: The voting ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$

The tally algorithm $\mathsf{TallyAlg}$

Input: a set of the voters $\mathcal{V}$, a set of the experts $\mathcal{E}$, a set of candidates $\mathcal{C}$, a table of ballots $\mathsf{ballots}$ and a table of voting power power.
Output: the tally result η.
Use n,m,ℓ to denote $|\mathcal{V}|,|\mathcal{E}|,|\mathcal{C}|$, respectively.

Init:

- For each candidate C_i, set initial score as $s_i := 0$. For each expert E_j, set initial score as $s_{\ell+j} := 0$.

Tally Computation:

- For each $\mathsf{V}_j \in \mathcal{V}$, let $v := \mathsf{ballots}[\mathsf{V}_j]$, set $s_v := s_v + \mathsf{power}[\mathsf{V}_i]$.
- For each $\mathsf{E}_i \in \mathcal{E}$, let $v := \mathsf{ballots}[\mathsf{E}_i]$, set $s_v := s_v + s_{\ell+i}$.

Output:

- Return $\eta := \{s_i\}_{i\in[1,\ell]}$.

Figure 12: The tally algorithm

The ballot-counting algorithm $\mathsf{CountAlg}$

Input: a set of the voters $\mathcal{V}$, a set of the experts $\mathcal{E}$, a set of candidates $\mathcal{C}$, a table of ballots $\mathsf{ballots}$.
Output: voters' ballot count bc_{V}.
Use n,m,ℓ to denote $|\mathcal{V}|,|\mathcal{E}|,|\mathcal{C}|$, respectively.

Init:

- For each candidate C_i, set initial ballot count as $c_i := 0$. For each expert E_j, set initial ballot count as $c_{\ell+j} := 0$.

Computation:

- For each $\mathsf{V}_j \in \mathcal{V}$, let $v := \mathsf{ballots}[\mathsf{V}_j]$, set $c_v := c_v + 1$.

Output:

- Return $\mathsf{bc}_{\mathsf{V}} := \{c_i\}_{i\in[1,\ell+m]}$.

Figure 13: The ballot-counting algorithm

Preparation phase. In the preparation phase, the ideal functionality $\mathcal{F}_{\text{vote}}^{n,m,\ell,t,k}$ receives the initialization commands from the trustees and the RA, and it sends initialization notification to the simulator $\mathcal{S}$. At the end of the preparation phase, $\mathcal{F}_{\text{vote}}^{n,m,\ell,t,k}$ sets state $:= 1$.
Registration phase. In the registration phase, the ideal functionality $\mathcal{F}_{\text{vote}}^{n,m,\ell,t,k}$ receives registration requests from voters and records their voting power. If there are t or more corrupted trustees, voting power will be leaked to the ideal world adversary $\mathcal{S}$. At the end of the registration phase, $\mathcal{F}_{\text{vote}}^{n,m,\ell,t,k}$ sets state $:= 2$.
Voting/Delegation phase. In the voting/delegation phase, the ideal functionality $\mathcal{F}_{\text{vote}}^{n,m,\ell,t,k}$ receives vote commands from voters and experts, and records their votes. Similarly, if there are t or more corrupted trustees, the votes will be leaked to $\mathcal{S}$. At the end of the voting/delegation phase, $\mathcal{F}_{\text{vote}}^{n,m,\ell,t,k}$ sets state $:= 3$.
Tally phase. In the tally phase, the ideal functionality $\mathcal{F}_{\text{vote}}^{n,m,\ell,t,k}$ receives tally commands from the trustees and performs the tally algorithm. The tally algorithm $\mathsf{TallyAlg}$ (Cf. Fig 12) takes as input all the ballots and voting power and outputs the final tally result. During the tally, experts' ballots and voters' ballot count are leaked (Cf. Fig. 13), but this does not affect ballot privacy of voters.
Ideal deception. The ideal deceiving strategy DI is controlled by the environment $\mathcal{Z}$. When a voter V_i is coerced by the ideal adversary $\mathcal{S}$ to vote for x, $\mathcal{Z}$ maps x to x' and instrusts DI_i to send $(\textsc{Vote}, \mathsf{sid}, x')$ to $\mathcal{F}_{\text{vote}}^{n,m,\ell,t,k}$. Note that, in the modeling, DI_i can either obey (i.e., $x = x'$) or deceive (i.e., $x \neq x'$).
Connection with the properties. It is easy to see that a protocol Π IUC-realizing $\mathcal{F}_{\text{vote}}^{n,m,\ell,t,k}$ has ballot privacy, verifiability and coercion-resistance. Firstly, voters' ballots are never leaked by $\mathcal{F}_{\text{vote}}^{n,m,\ell,t,k}$ if the shuffler and majority of trustees are honest. Secondly, nobody can falsify the final tally result computed by $\mathcal{F}_{\text{vote}}^{n,m,\ell,t,k}$. Thirdly, the definition of the ideal deception DI ensures that a voter can cast-as-intend even if he is being coerced.
The real world execution. In the real world, we invoke the distributed key generation functionality $\mathcal{F}_{\text{DKG}}^{t,k}[\mathbb{G}]$ (Cf. Fig. 1) and the secure channel functionality $\mathcal{F}_{\text{sc}}$ (Cf. Fig. 3). The parties perform the voting protocol. When a voter V_i is coerced, he switches to the real deceiving strategy DR_i to resist coercion.

5 The Protocol

In this section, we give a detailed protocol description of our voting scheme and formally prove security of our scheme in the UC framework.

5.1 Protocol Description

Coercion-resistant voting $\Pi_{\text{vote}}^{n,m,\ell,t,k}$

Denote the voters as $\mathcal{V} := \{\mathsf{V}_1, \dots, \mathsf{V}_n\}$, the experts as $\mathcal{E} := \{\mathsf{E}_1, \dots, \mathsf{E}_m\}$, the candidates as $\mathcal{C} := \{\mathsf{C}_1, \dots, \mathsf{C}_\ell\}$, the registration authority as RA, the trustees as $\mathcal{T} := \{\mathsf{T}_1, \dots, \mathsf{T}_k\}$. The protocol is parameterized with threshold t.

Preparation Phase:

Upon receiving $(\textsc{Init}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the trustee T_i does the following:

∘ Send $(\textsc{Keygen}, \mathsf{sid})$ to $\mathcal{F}_{\text{DKG}}^{t,k}$ and receive $(\textsc{Keygen}, \mathsf{sid}, (\mathsf{pk}_\mathsf{T}, \mathsf{sk}_{\mathsf{T},i}))$.

∘ Send $(\textsc{Write}, \mathsf{sid}, \mathsf{pk}_\mathsf{T})$ to $\mathcal{G}_{\text{PBB}}$.

Upon receiving $(\textsc{Init}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the RA does the following:

∘ Generate a public-private signing key-pair $(\mathsf{pk}_\mathsf{RA}, \mathsf{sk}_\mathsf{RA}) \leftarrow \mathsf{Sig.Keygen}(1^\lambda)$.

∘ Send $(\textsc{Write}, \mathsf{sid}, \mathsf{pk}_\mathsf{RA})$ to $\mathcal{G}_{\text{PBB}}$.

Registration Phase:

Upon receiving $(\textsc{Reg}, \mathsf{sid}, \alpha_j)$ from the environment $\mathcal{Z}$, the voter V_j does the following:

∘ Generate a public-private signing key-pair $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{Sig.Keygen}(1^\lambda)$.

∘ Send $(\textsc{Send}, \mathsf{sid}, \mathsf{RA}, \langle K_j, A_j, \mathsf{tx}_j, \delta_j \rangle)$ to $\mathcal{F}_{\text{sc}}$, where $K_j \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_\mathsf{T}}(\mathsf{pk}_j)$, $A_j \leftarrow \mathsf{LE.Enc}_{\mathsf{pk}_\mathsf{T}}(\alpha_j)$, tx_j is the transaction that freezes the voter's stake, and $\delta_j \leftarrow \mathsf{NIZK}_{\text{power}}.\mathsf{Prove}(A_j, \mathsf{tx}_j)$ is a NIZK proof that the frozen stake equals α_j (Cf. Appendix A).

Upon receiving $(\textsc{Sent}, \mathsf{sid}, \mathsf{V}_j, \langle K_j, A_j, \mathsf{tx}_j, \delta_j \rangle)$ from $\mathcal{F}_{\text{sc}}$, the registration authority RA does the following:

∘ Compute $b := \mathsf{NIZK}_{\text{power}}.\mathsf{Verify}(\delta_j, A_j || \mathsf{tx}_j)$. If $b = 0$, send $(\textsc{Send}, \mathsf{sid}, \mathsf{V}_j, \mathsf{REJECT})$ to $\mathcal{F}_{\text{sc}}$.

∘ Else, compute $\tilde{K}_j \leftarrow \mathsf{EC.Rand}_{\mathsf{pk}_\mathsf{T}}(K_j)$. Send $(\textsc{Write}, \mathsf{sid}, \langle \tilde{K}_j, A_j, \mathsf{tx}_j, \delta_j, \sigma_{\mathsf{RA},j} \rangle)$ to $\mathcal{G}_{\text{PBB}}$, where $\sigma_{\mathsf{RA},j} \leftarrow \mathsf{Sig.sign}_{\mathsf{sk}_\mathsf{RA}}(\tilde{K}_j || A_j || \mathsf{tx}_j || \delta_j)$.

∘ Generate a designated verifier proof π_{DVP} of re-encryption correctness (Cf. Appendix A) and send $(\textsc{Send}, \mathsf{sid}, \mathsf{V}_j, \pi_{\mathsf{DVP}})$ to $\mathcal{F}_{\text{sc}}$.

Voting/Delegation Phase:

Upon receiving $(\textsc{Vote}, \mathsf{sid}, v_i)$ from the environment $\mathcal{Z}$, the expert E_i does the following:

∘ Compute $\boldsymbol{c}_i \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_\mathsf{T}}(v_i)$ and the corresponding NIZK $\tau_i \leftarrow \mathsf{NIZK}_{\text{knowledge}}.\mathsf{Prove}(\boldsymbol{c}_i)$, which is a proof of plaintext knowledge of $\boldsymbol{c}_i$ (Cf. Fig. 18).

∘ Compute $\sigma_{\mathsf{E}_i} \leftarrow \mathsf{Sig.sign}_{\mathsf{sk}_{\mathsf{E}_i}}(\boldsymbol{c}_i)$.

∘ Denote the ballot as $B_i^\mathsf{E} := (\mathsf{E}_i, \boldsymbol{c}_i, \tau_i, \sigma_{\mathsf{E}_i})$.

∘ Send $(\textsc{Write}, \mathsf{sid}, B_i^\mathsf{E})$ to $\mathcal{G}_{\text{PBB}}$.

Upon receiving $(\textsc{Vote}, \mathsf{sid}, v_j)$ from the environment $\mathcal{Z}$, the voter V_j does the following:

∘ Compute $\boldsymbol{u}_j \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_\mathsf{T}}(v_j)$ and the corresponding NIZK $\pi_i \leftarrow \mathsf{NIZK}_{\text{knowledge}}.\mathsf{Prove}(\boldsymbol{u}_i)$, which is a proof of plaintext knowledge of $\boldsymbol{u}_i$ (Cf. Fig. 18).

∘ Compute $\sigma_j \leftarrow \mathsf{Sig.sign}_{\mathsf{sk}_j}(\boldsymbol{u}_j)$.

∘ Denote the ballot as $B_j = (\mathsf{pk}_j, \boldsymbol{u}_j, \pi_j, \sigma_j)$.

∘ Send $(\textsc{Write}, \mathsf{sid}, B_j)$ to $\mathcal{G}_{\text{PBB}}$.

Upon coerced, the voter V_j switches to the real deception strategy as described in Fig. 14.

Real Deceiving Strategy DR

The real deception strategy DR_i internally runs DI_i. Upon coerced, it does the following:

- Generate a fake signing key pair: $\mathsf{sk}'_j, \mathsf{pk}'_j \leftarrow \mathsf{Sig.Keygen}(1^\lambda)$.
- Compute $K'_j \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_\mathsf{T}}(\mathsf{pk}'_j)$.
- Send $(\textsc{Write}, \mathsf{sid}, \langle K'_j, A_0, \rho_j \rangle)$ to $\mathcal{G}_{\mathsf{PBB}}$, where $A_0 = \mathsf{LE.Enc}_{\mathsf{pk}_\mathsf{T}}(0;0)$ and $\rho_j \leftarrow \mathsf{NIZK}_{\mathsf{sk}}.\mathsf{Prove}(K'_j)$ is a NIZK proof of knowledge of sk'_j (Cf. Fig. 20).
- Use $(\mathsf{sk}'_j, \mathsf{pk}'_j)$ to perform the voting procedure, following the coercer's instruction.
- Use $(\mathsf{sk}_j, \mathsf{pk}_j)$ to perform the voting procedure again, where the candidate v_i is the same as the one submitted by DI_i.

Figure 14: Real Deceiving Strategy DR

Tally Phase:

Upon receiving $(\textsc{VoteEnd}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the shuffler does the following:

∘ Send $(\textsc{Read}, \mathsf{sid})$ to $\mathcal{G}_{\mathsf{PBB}}$ and get all the ballots and "encrypted public key items".

∘ For each voter's ballot $B_j = (\mathsf{pk}_j, \boldsymbol{u}_j, \pi_j, \sigma_j)$, check (i) $\mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Verify}(\pi_j, \boldsymbol{u}_j) \stackrel{?}{=} 1$; (ii) $\mathsf{Sig.Verify}_{\mathsf{pk}_j}(\sigma_j, \boldsymbol{u}_j) \stackrel{?}{=} 1$. Remove the invalid ballots and remove π_j, σ_j. Now a voter's ballot β is of the form $(K, \boldsymbol{u})$.

∘ For each "encrypted public key item" $\langle \tilde{K}_j, A_j, \mathsf{tx}_j, \delta_j, \sigma_{\mathsf{RA},j} \rangle$ published by RA, check (i) $\mathsf{NIZK}_{\mathsf{power}}.\mathsf{Verify}(\delta_j, A_j || \mathsf{tx}_j) \stackrel{?}{=} 1$; (ii) $\mathsf{Sig.Verify}_{\mathsf{pk}_{\mathsf{RA}}}(\sigma_{\mathsf{RA},j}, \tilde{K}_j || A_j || \mathsf{tx}_j || \delta_j) \stackrel{?}{=} 1$. Remove the invalid ones and remove $\mathsf{tx}_j, \delta_j, \sigma_{\mathsf{RA},j}$.

∘ For each "encrypted public key item" $\langle K'_j, A_0, \rho_j \rangle$ published by a voter, check $\mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Verify}(\rho_j, K'_j) \stackrel{?}{=} 1$. Remove the invalid ones and remove ρ_j.

∘ Put all the valid "encrypted public key items" together. At this point, each item W is of the form (K, A), where K is the encrypted public key, A is the encrypted voting power.

∘ Verifiably shuffle re-encrypt the encrypted public key items (Cf. [5]).

∘ Send $(\textsc{Write}, \mathsf{sid}, \langle \{\beta\}, \{W\}, \pi \rangle)$ to $\mathcal{G}_{\mathsf{PBB}}$, where π is the proof of shuffle correctness (Cf. [5]).

Upon receiving $(\textsc{Tally}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the trustee $\mathsf{T}_t, t \in [k]$ does the following:

∘ Send $(\textsc{Read}, \mathsf{sid})$ to $\mathcal{G}_{\mathsf{PBB}}$ to get $\{\beta\}, \{W\}$.

∘ Jointly decrypt the public keys (i.e. K) in $\{W\}$ (Cf. [20]).

∘ For each ballot β, if the public key does not match any public key in $\{W\}$, drop it.

∘ Put the corresponding A together with $\boldsymbol{u}$, i.e., assume $\beta = (K_B, \boldsymbol{u})$ and $W = (K_W, A)$, and K_B, K_A are decrypted to the same public key, then we put A and $\boldsymbol{u}$ together to form a new item $I := (A, \boldsymbol{u})$.

∘ Send $(\textsc{Write}, \mathsf{sid}, \{I\})$ to $\mathcal{G}_{\mathsf{PBB}}$.

∘ Send $(\textsc{Shuffle}, \mathsf{sid})$ to the shuffler.

Upon receiving $(\textsc{Shuffle}, \mathsf{sid})$ from the trustees, the shuffler does the following:

∘ Send $(\textsc{Read}, \mathsf{sid})$ to $\mathcal{G}_{\mathsf{PBB}}$ and get $\{I\}$.

∘ Shuffle {I} and send $(\textsc{Write}, \mathsf{sid}, \langle \{I'\}, \pi \rangle)$ to $\mathcal{G}_{\mathsf{PBB}}$, where π is the proof of shuffle correctness (Cf. [5]).

∘ Send $(\textsc{ShuffleEnd}, \mathsf{sid})$ to all the trustees.

Upon receiving $(\textsc{ShuffleEnd}, \mathsf{sid})$ from the shuffler, the trustee $\mathsf{T}_t, t \in [k]$ does the following:

∘ For each candidate C_i, set initial score as $\boldsymbol{s}_i := \mathsf{LE.Enc}_{\mathsf{pk}_\mathsf{T}}(0)$. For each expert E_j, set initial score as $\boldsymbol{s}_{\ell+j} := \mathsf{LE.Enc}_{\mathsf{pk}_\mathsf{T}}(0)$.

∘ For each item $I := (A, \boldsymbol{u})$, jointly decrypt $\boldsymbol{u}$ to v (Cf. [20]) and update candidate (or expert) v's score $\boldsymbol{s}_v := \boldsymbol{s}_v \cdot A$.

∘ After tallying all the voters' ballots, for each expert's ballot $B_i^\mathsf{E} := (\mathsf{E}_i, \boldsymbol{c}_i, \tau_i, \sigma_{\mathsf{E}_i})$, check (i) $\mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Verify}(\tau_i, \boldsymbol{c}_i) \stackrel{?}{=} 1$; (ii) $\mathsf{Sig.Verify}_{\mathsf{pk}_{\mathsf{E}_i}}(\sigma_{\mathsf{E}_i}, \boldsymbol{c}_i) \stackrel{?}{=} 1$. Remove the invalid ballots and $\tau_i, \sigma_{\mathsf{E}_i}$. Now an expert's ballot β^E is of the form $(\mathsf{E}_i, \boldsymbol{c}_i)$.

∘ Form a list of each expert's encrypted score and encrypted candidate, i.e., expert E_i's entry is of the form $(\boldsymbol{s}_{\ell+i}, \boldsymbol{c}_i)$.

∘ For each expert's entry $(\boldsymbol{s}, \boldsymbol{c})$, jointly decrypt $\boldsymbol{c}$ to v (Cf. [20]) and update candidate v's score $\boldsymbol{s}_v := \boldsymbol{s}_v \cdot \boldsymbol{s}$.

∘ For each candidate C_i, jointly decrypt the score $\boldsymbol{s}_i$ to s_i (Cf. [20]).

∘ Send $(\textsc{Write}, \mathsf{sid}, \{s_i\}_{i \in [\ell]})$ to $\mathcal{G}_{\mathsf{PBB}}$.

Upon receiving $(\textsc{ReadTally}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the party P does the following:

∘ Send $(\textsc{Read}, \mathsf{sid})$ to $\mathcal{G}_{\mathsf{PBB}}$ and get $\{s_i\}_{i \in [\ell]}$.

∘ Return $(\textsc{ReadTallyReturn}, \mathsf{sid}, \{s_i\}_{i \in [\ell]})$ to the requestor.

5.2 Complexity

In the *preparation phase*, the RA takes $O(1)$ time to generate the signing key pair, and the trustees take $O(k)$ time to perform the DKG protocol, where k is the number of trustees. In the *registration phase*, a voter generates a NIZK proof of voting power correctness and verifies a designated verifier proof, which has $O(1)$ complexity. In the *voting/delegation phase*, a voter encrypts his choice and generates a NIZK proof of plaintext knowledge, which has $O(1)$ complexity, too. In the *tally phase*, the shuffler shuffles the ballots and the encrypted public key items, which has $O(n)$ complexity (counting cryptographic operations only), where n is the number of voters. Then, the trustees decrypt the public keys, do the matching between voting power and candidate, and decrypt

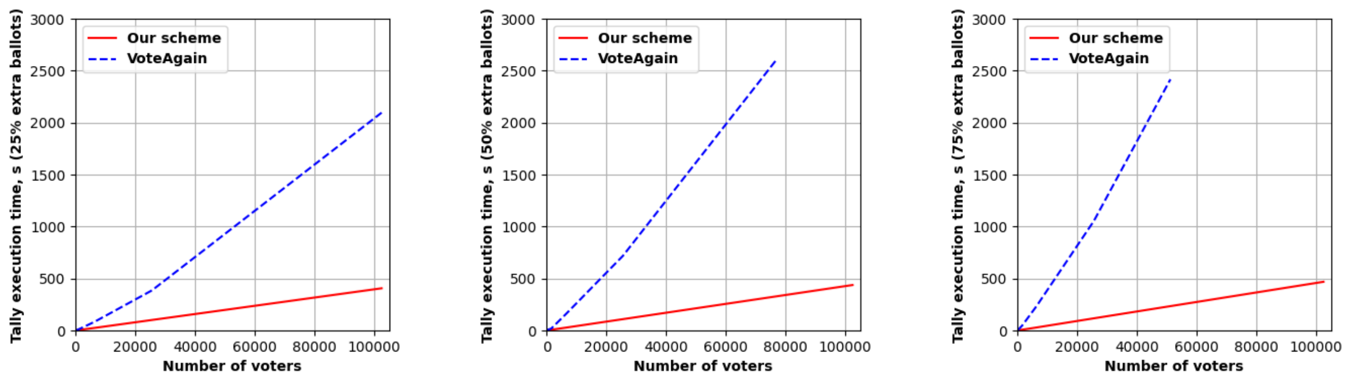

Figure 15: Comparison of tally execution time between our scheme and VoteAgain [26] (with extra ballot rate as 25%, 50%, 75% from left to right). In VoteAgain, $x\%$ extra ballot rate means that $x\%$ voters re-vote once; in our scheme, $x\%$ extra ballot rate means that $x\%$ voters cast one fake ballot.

the candidate. Thus, the time complexity of a trustee is also $O(n)$. Adding them together, the whole scheme has $O(n)$ time complexity.

5.3 Security

We show the security of our construction via the following theorem.

Theorem 1. *Assume that the NIZKs* $\mathsf{NIZK}_i, i \in \{\mathsf{power}, \mathsf{knowledge}, \mathsf{DVF\text{-}reenc}, \mathsf{sk}, \mathsf{shuffle}, \mathsf{Dec}\}$ *are complete, sound, and zero-knowledge. Assume that the ElGamal encryption scheme* EC *is IND-CPA secure. Assume that* Sig *is a signature scheme satisfying EUF-CMA. Then the protocol* $\Pi_{\mathsf{vote}}^{n,m,\ell,t,k}$ *IUC-realizes* $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$ *against static corruption and adaptive coercion in the* $\{\mathcal{F}_{\mathsf{DKG}}^{t,k}[\mathbb{G}], \mathcal{F}_{\mathsf{sc}}, \mathcal{G}_{\mathsf{PBB}}\}$*-hybrid world.*

The proof of the theorem is deferred to Appendix C.

6 Implementation and Evaluation

We implement a prototype of our voting scheme in Rust. The implementation uses OpenSSL 1.1.1t to provide the basic elliptic curve math and it uses Schnorr signature as the signature scheme. We evaluated all the cryptographic building blocks and the time consumption in each phase. The experiments are performed on a workstation with Intel Core i7-1165G7 @2.80GHz and 32GB RAM running Ubuntu 20.04.4 LTS x64, using the elliptic curve secp256r1.

Preparation phase. We evaluate the DKG execution time and traffic with respect to different numbers of trustees: from 4 to 64. Results are given in Fig. 16.

Registration phase. In the registration phase, the voter uses $\mathsf{NIZK}_{\mathsf{power}}$ in the registration request to prove that the frozen stake is equal to the encrypted voting power, and the RA proves re-encryption correctness by a designated verifier

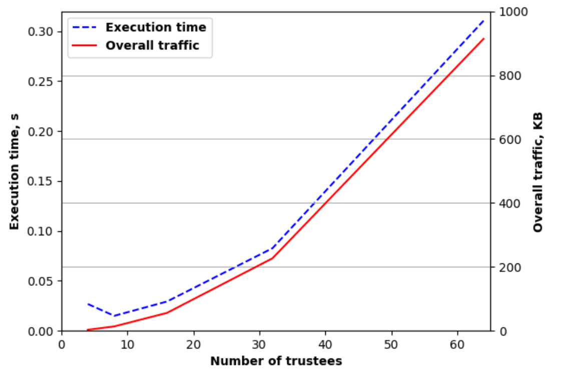

Figure 16: DKG execution time and overall traffic with respect to different numbers of trustees

proof $\mathsf{NIZK}_{\mathsf{DVF\text{-}reenc}}$. Generating a registration request costs 430.95 µs and its size is 475 bytes. The designated verifier proof costs 102.91 µs to generate and 181.69 µs to verify, and its size is 141 bytes. Generating a fake "encrypted public key item" costs 336.05 µs and the size is 267 bytes.

Voting/Delegation phase. In the voting/delegation phase, voters and experts encrypt the choice and sign it and using $\mathsf{NIZK}_{\mathsf{knowledge}}$ to prove plaintext knowledge of the ballot. It takes 283.27 µs to generate a ballot. The size of a voter's ballot is 358 bytes and the size of an expert's ballot is 332 bytes.

Tally phase. We evaluate the tally execution time with respect to different numbers of voters and different extra ballot rates and compare the results with VoteAgain [26]. Here, extra ballot rate represents how many voters cast extra ballots, i.e., in VoteAgain, 50% extra ballot rate means that 50% voters re-vote once; in our scheme, 50% extra ballot rate means that 50% voters cast one fake ballot. Note that, in our scheme, a voter's ballot takes more time to tally than an expert's ballot (because voters' ballots are shuffled and experts' ballots are not shuffled), so we set expert number as zero in the experiments.

Fig. 15 shows the tally execution time compared with VoteAgain [26] when the extra ballot rates are 25%, 50%, and 75% from left to right. VoteAgain's benchmark fails in 102400 voters with 50% and 75% extra ballot rate (probably because of too large ciphertext input). We can see that our scheme's execution time grows linearly and VoteAgain's execution time grows quasi-linearly ($O(n \log n)$). A higher rate of extra ballots confers a greater advantage, as it necessitates VoteAgain to introduce a substantial quantity of dummy ballots in this scenario. In large-scale voting with more than 10000 voters and over 50% extra ballot rate, our scheme's tally execution time is over 6x faster than VoteAgain.

7 Conclusion

In this work, we propose the first scalable coercion-resistant blockchain decision-making scheme that supports differential voting power and liquid democracy. It is scalable in the sense that it has constant ballot size and linear complexity. We formally prove the scheme secure under the UC incoercibility framework without any extra strong assumptions. Compared with existing voting schemes, our scheme has an advantage over all of them, so it is suitable to be applied to large-scale coercion-resistant blockchain voting programs.

References

[1] Dirk Achenbach, Carmen Kempka, Bernhard Löwe, and Jörn Müller-Quade. Improved coercion-resistant electronic elections through deniable re-voting. {*USENIX*} *Journal of Election Technology and Systems* ({*JETS*}), 3:26–45, 2015.

[2] Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security 2008: 17th USENIX Security Symposium*, pages 335–348, San Jose, CA, USA, July 28 – August 1, 2008. USENIX Association.

[3] Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. Incoercible multi-party computation and universally composable receipt-free voting. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 763–780, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[4] Roberto Araújo, Amira Barki, Solenn Brunet, and Jacques Traoré. Remote electronic voting can be efficient, verifiable and coercion-resistant. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 224–232, Christ Church, Barbados, February 26, 2016. Springer, Heidelberg, Germany.

[5] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[6] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press.

[7] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[8] Sergiu Bursuc, Gurchetan S Grewal, and Mark D Ryan. Trivitas: Voters directly verifying votes. In *International Conference on E-Voting and Identity*, pages 190–207. Springer, 2011.

[9] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

[10] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany.

[11] David Chaum. Random-sample voting. *White Paper*, 2016.

[12] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[13] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

[14] Jeremy Clark and Urs Hengartner. Selections: Internet voting with over-the-shoulder coercion-resistance. In George Danezis, editor, *FC 2011: 15th International Conference on Financial Cryptography and Data Security*, volume 7035 of *Lecture Notes in Computer Science*, pages 47–61, Gros Islet, St. Lucia, February 28 – March 4, 2012. Springer, Heidelberg, Germany.

[15] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy*, pages 354–368, Oakland, CA, USA, May 18–21, 2008. IEEE Computer Society Press.

[16] Véronique Cortier and Joseph Lallemand. Voting: You can't have privacy without individual verifiability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 53–66, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[17] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Heidelberg, Germany.

[18] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[19] Bryan Alexander Ford. Delegative democracy. Technical report, 2002.

[20] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

[21] Kristian Gjøsteen. The norwegian internet voting protocol. In *E-Voting and Identity: Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers 3*, pages 1–18. Springer, 2012.

[22] Lucca Hirschi, Lara Schmid, and David A. Basin. Fixing the achilles heel of E-voting: The bulletin board. In Ralf Küsters and Dave Naumann, editors, *CSF 2021: IEEE 34th Computer Security Foundations Symposium*, pages 1–17, Virtual Conference, June 21–24, 2021. IEEE Computer Society Press.

[23] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 539–556, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.

[24] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 61–70, 2005.

[25] Philipp Locher, Rolf Haenni, and Reto E. Koenig. Coercion-resistant internet voting with everlasting privacy. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 161–175, Christ Church, Barbados, February 26, 2016. Springer, Heidelberg, Germany.

[26] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. VoteAgain: A scalable coercion-resistant voting system. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020: 29th USENIX Security Symposium*, pages 1553–1570. USENIX Association, August 12–14, 2020.

[27] Emmanouil Magkos, Mike Burmester, and Vassilis Chrissikopoulos. Receipt-freeness in large-scale elections without untappable channels. *Towards the E-Society: E-Commerce, E-Business, and E-Government*, pages 683–693, 2001.

[28] Peter YA Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. Prêt à voter: a voter-verifiable voting system. *IEEE transactions on information forensics and security*, 4(4):662–673, 2009.

[29] Snapshot. Snapshot. Online: https://snapshot.org (Last accessed: 2023-04-02).

[30] Douglas Wikström. Universally composable dkg with linear number of exponentiations. In *Security in Communication Networks: 4th International Conference, SCN 2004, Amalfi, Italy, September 8-10, 2004, Revised Selected Papers 4*, pages 263–277. Springer, 2005.

[31] Bingsheng Zhang, Roman Oliynykov, and Hamed Balogun. A treasury system for cryptocurrencies: Enabling better collaborative intelligence. In *ISOC Network and Distributed System Security Symposium – NDSS 2019*, San Diego, CA, USA, February 24–27, 2019. The Internet Society.

A NIZKs

In this section, we show the construction of the NIZKs used in our system. There are six zero-knowledge proofs in our scheme for proving: (i) voting power correctness ($\mathsf{NIZK}_{\mathsf{power}}$); (ii) ElGamal encryption plaintext

Sigma protocol for voting power correctness

CRS: g, h, m.
Statement: $A = (A_1, A_2), v = (v_1, v_2)$.
Witness: α, r_1, r_2 such that $A = (g^{r_1}, g^{\alpha} h^{r_1}) \wedge v = (g^{r_2}, g^{\alpha} m^{r_2})$.

Prover:

- Pick random $\alpha', r'_1, r'_2 \leftarrow \mathbb{Z}_q$;
- Compute $a_1 := g^{r'_1}, a_2 := g^{\alpha'} h^{r'_1}, a_3 := g^{r'_2}, a_4 := g^{\alpha'} m^{r'_2}$;
- $P \rightarrow V$: a_1, a_2, a_3, a_4.

Verifier:

- $V \rightarrow P$: random $e \leftarrow \mathbb{Z}_q$.

Prover:

- Compute $z_1 := r'_1 + e \cdot r_1, z_2 := r'_2 + e \cdot r_2, z_3 := \alpha' + e \cdot \alpha$;
- $P \rightarrow V$: z_1, z_2, z_3.

Verifier:

- Output 1 if and only if the following holds:
  - $g^{z_1} = a_1 \cdot A_1^e$;
  - $g^{z_3} h^{z_1} = a_2 \cdot A_2^e$;
  - $g^{z_2} = a_3 \cdot v_1^e$;
  - $g^{z_3} h^{z_2} = a_4 \cdot v_2^e$.

Figure 17: Sigma protocol for voting power correctness

knowledge ($\mathsf{NIZK}_{\mathsf{knowledge}}$); (iii) re-encryption correctness ($\mathsf{NIZK}_{\mathsf{DVF\text{-}reenc}}$); (iv) knowledge of secret key ($\mathsf{NIZK}_{\mathsf{sk}}$) (v) shuffle correctness ($\mathsf{NIZK}_{\mathsf{shuffle}}$); and (vi) decryption correctness ($\mathsf{NIZK}_{\mathsf{Dec}}$). We adopt Bayer and Groth's scheme [5] for shuffle correctness and Genarro *et al.*'s scheme [20] for decryption correctness. Here, we will demonstrate how to use Sigma protocols to construct the other four zero-knowledge proofs. In practice, they will be transformed into NIZKs by Fiat-Shamir heuristic [18].

Proof of voting power correctness. In the registration phase, the voter will create a transaction that freezes some stake. Then, he sends the transaction tx, encrypted voting power A, and proves that the encrypted voting power is the same as the value of tx. In a privacy-preserving blockchain cryptocurrency system, tx usually contains an encrypted transaction value v. In this case, the zero-knowledge proof proves that A and v encrypt the same value. If the transaction value is encrypted by Lifted Elgamal, then Fig. 17 shows the Sigma protocol for voting power correctness. If it is encrypted by a hybrid encryption scheme (e.g., ZCash), then we can utilize zkSNARK to construct the zero-knowledge proof.

Proof of ElGamal encryption plaintext knowledge. In the voting phase, we use a NIZK for ballot plaintext knowledge to prevent copying the other voter's choice. This can also be proven with Sigma protocol, depicted in Fig. 18.

Proof of re-encryption correctness. In the registration phase, the RA needs to generate a designated verifier proof

Sigma protocol for plaintext knowledge

CRS: g, h.
Statement: $c = (c_1, c_2)$.
Witness: m, r such that $c_1 = g^r \wedge c_2 = m \cdot h^r$.

Prover:

- Pick random $r' \leftarrow \mathbb{Z}_q, m' \leftarrow \mathbb{G}$;
- Compute $a_1 := g^{r'}, a_2 := m' \cdot h^{r'}$;
- $P \rightarrow V$: a_1, a_2.

Verifier:

- $V \rightarrow P$: random $e \leftarrow \mathbb{Z}_q$.

Prover:

- Compute $z_1 := r' + e \cdot r, z_2 := m' \cdot m^e$;
- $P \rightarrow V$: z_1, z_2.

Verifier:

- Output 1 if and only if the following holds:
  - $g^{z_1} = a_1 \cdot c_1^e$;
  - $z_2 \cdot h^{z_1} = a_2 \cdot c_2^e$.

Figure 18: Sigma protocol for ElGamal encryption plaintext knowledge

Sigma protocol for re-encryption correctness

CRS: g, h.
Statement: $u = (u_1, u_2), v = (v_1, v_2)$.
Witness: r such that $v_1 = u_1 \cdot g^r \wedge v_2 = u_2 \cdot h^r$.

Prover:

- Pick random $r' \leftarrow \mathbb{Z}_q$;
- Compute $a_1 = g^{r'}, a_2 := h^{r'}$;
- $P \rightarrow V$: a_1, a_2.

Verifier:

- $V \rightarrow P$: random $e \leftarrow \mathbb{Z}_q$.

Prover:

- Compute $z := r' + e \cdot r$;
- $P \rightarrow V$: z.

Verifier:

- Output 1 if and only if the following holds:
  - $g^z = a_1 \cdot (v_1/u_1)^e$;
  - $h^z = a_2 \cdot (v_2/u_2)^e$.

Figure 19: Sigma protocol for re-encryption correctness

Sigma protocol for knowledge of secret key

CRS: g, h.
Statement: $c = (c_1, c_2)$.
Witness: x, r such that $c_1 = g^r \wedge c_2 = g^x \cdot h^r$.

Prover:

- Pick random $r' \leftarrow \mathbb{Z}_q, x' \leftarrow \mathbb{G}$;
- Compute $a_1 := g^{r'}, a_2 := g^{x'} \cdot h^{r'}$;
- $P \rightarrow V$: a_1, a_2.

Verifier:

- $V \rightarrow P$: random $e \leftarrow \mathbb{Z}_q$.

Prover:

- Compute $z_1 := r' + e \cdot r, z_2 := x' + e \cdot x$;
- $P \rightarrow V$: z_1, z_2.

Verifier:

- Output 1 if and only if the following holds:
  - $g^{z_1} = a_1 \cdot c_1^e$;
  - $g^{z_2} \cdot h^{z_1} = a_2 \cdot c_2^e$.

Figure 20: Sigma protocol for knowledge of secret key

for re-encryption correctness. To make it a designated verifier proof, the statement is "this is a correct re-encryption OR I know the verifier's secret key" so that the verifier can simulate the proof. The Sigma protocol for re-encryption correctness is depicted in Fig. 19. By the CDS composition [17], we can compose the Sigma protocol for re-encryption correctness and the standard Schnorr protocol to construct the designated verifier proof of re-encryption correctness.
Proof of knowledge of secret key. To publish an (encrypted) fake voting public key on the PBB, the voter needs to prove knowledge of the corresponding secret key. This is a variant of the Schnorr protocol, depicted in Fig. 20.

B Security Definitions of NIZK, Encryption, and Signature

Here, we give formal game-based definitions of completeness, soundness, zero-knowledge of a NIZK, IND-CPA property of an encryption scheme, and EUF-CMA property of a signature scheme.
NIZK. A non-interactive zero-knowledge proof (NIZK) for relation $\mathcal{R}$ has four PPT algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ such that
$(\sigma, \tau) \leftarrow \mathsf{Setup}(\mathcal{R})$: The setup algorithm outputs a common reference string σ and a simulation trapdoor τ for relation $\mathcal{R}$.
$\pi \leftarrow \mathsf{Prove}(\mathcal{R}, \sigma, \phi, w)$: the prover algorithm takes as input a common reference string σ and $(\phi, w) \in \mathcal{R}$ and outputs a proof π.
$0/1 \leftarrow \mathsf{Verify}(\mathcal{R}, \sigma, \phi, \pi)$: the verification algorithm takes as input a common reference string σ, a statement ϕ and a proof π, and it returns 0 or 1 for rejection or acceptance, respectively.
$\pi \leftarrow \mathsf{Sim}(\mathcal{R}, \tau, \phi)$: the simulation algorithm takes as input a simulation trapdoor τ and a statement ϕ, and it outputs a proof π.
Completeness. A NIZK protocol is complete if an honest prover can always successfully convince an honest verifier. Formally, for all $(\phi, w) \in \mathcal{R}$,

$$\Pr[(\sigma, \tau) \leftarrow \mathsf{Setup}(\mathcal{R}); \pi \leftarrow \mathsf{Prove}(\mathcal{R}, \sigma, \phi, w) : \mathsf{Verify}(\mathcal{R}, \sigma, \phi, \pi) = 1] = 1$$

Zero-knowledge. A proof is zero-knowledge if no other information is leaked except that the statement is true. Consider the following experiment:
Experiment $\mathrm{EXPT}^{\mathsf{zk}}_{\mathcal{A},\mathsf{NIZK}}(\lambda)$:

1. For a relation $\mathcal{R}$, $(\sigma, \tau) \leftarrow \mathsf{Setup}(\mathcal{R})$, $(\phi, w) \in \mathcal{R}$, the challenger computes $\pi_0 \leftarrow \mathsf{Prove}(\mathcal{R}, \sigma, \phi, w)$ and $\pi_1 \leftarrow \mathsf{Sim}(\mathcal{R}, \tau, \phi)$.
2. The challenger picks a random bit $b \in \{0, 1\}$.
3. $\mathcal{A}$ is given (σ, π_b) as input, and it outputs a guess bit $b' \in \{0, 1\}$.
4. If $b = b'$, output 1; otherwise, output 0.

A NIZK is zero-knowledge if the adversary $\mathcal{A}$'s advantage $\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}}(\mathcal{A}, \lambda) := |2 \cdot \Pr[\mathrm{EXPT}^{\mathsf{zk}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) = 1] - 1|$ is negligible in λ.
Soundness. A proof is sound if it is not possible for a prover to prove a false statement. Consider the following experiment:
Experiment $\mathrm{EXPT}^{\mathsf{sound}}_{\mathcal{A},\mathsf{NIZK}}(\lambda)$:

1. For a relation $\mathcal{R}$, $(\sigma, \tau) \leftarrow \mathsf{Setup}(\mathcal{R})$.
2. Given σ as input, $\mathcal{A}$ outputs (ϕ, π).
3. If $\mathsf{Verify}(\mathcal{R}, \sigma, \phi, \pi) = 1$ and $\phi \notin L_{\mathcal{R}}$, output 1; otherwise, output 0.

A NIZK is sound if the adversary $\mathcal{A}$'s advantage $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{NIZK}}(\mathcal{A}, \lambda) := \Pr[\mathrm{EXPT}^{\mathsf{sound}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) = 1]$ is negligible in λ.
Encryption scheme. An encryption scheme consists of three PPT algorithms $(\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$. The ElGamal encryption scheme we used is IND-CPA secure. Formally, consider the following IND-CPA experiment:
Experiment $\mathrm{EXPT}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{EC}}(\lambda)$:

1. The challenger performs the key generation algorithm $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Keygen}(\lambda)$ and sends pk to the adversary $\mathcal{A}$.
2. $\mathcal{A}$ sends m_0, m_1 to the challenger.
3. The challenger picks a random bit $b \in \{0, 1\}$ and sends $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b)$ to $\mathcal{A}$.

4. $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. If $b = b'$, output 1; otherwise, output 0.

An encryption scheme is IND-CPA secure if the adversary $\mathcal{A}$'s advantage $\mathsf{Adv}_{\mathsf{EC}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A},\lambda) := |2 \cdot \Pr[\mathrm{EXPT}_{\mathcal{A},\mathsf{EC}}^{\mathsf{IND\text{-}CPA}}(\lambda) = 1] - 1|$ is negligible in λ.

Signature. A signature scheme consists of three PPT algorithms $(\mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify})$. We require the underlying signature scheme to be existentially unforgeable under chosen message attack (EUF-CMA). The EUF-CMA experiment is as follows:

Experiment $\mathrm{EXPT}_{\mathcal{A},\mathsf{Sig}}^{\mathsf{EUF\text{-}CMA}}(\lambda)$:

1. The challenger performs the key generation algorithm $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Keygen}(\lambda)$ and sends pk to the adversary $\mathcal{A}$.
2. $\mathcal{A}$ can repeatedly request for signatures on chosen messages $(m_0,\ldots,m_q)$, and receives the valid signatures $(\sigma_0,\ldots,\sigma_q)$ in response.
3. $\mathcal{A}$ outputs a message and signature (m^*,σ^*).
4. If m^* is not one of the messages requested in step 2, and $\mathsf{Verify}_{\mathsf{pk}}(m^*,\sigma^*) = 1$, output 1; otherwise, output 0.

A signature scheme is EUF-CMA if the adversary $\mathcal{A}$'s advantage $\mathsf{Adv}_{\mathsf{Sig}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A},\lambda) := \Pr[\mathrm{EXPT}_{\mathcal{A},\mathsf{Sig}}^{\mathsf{EUF\text{-}CMA}}(\lambda) = 1]$ is negligible in λ.

C Proof of Theorem 1

Theorem 1 *Assume that the NIZKs* $\mathsf{NIZK}_i, i \in \{\mathsf{power}, \mathsf{knowledge}, \mathsf{DVF\text{-}reenc}, \mathsf{sk}, \mathsf{shuffle}, \mathsf{Dec}\}$ *are complete, sound, and zero-knowledge. Assume that the ElGamal encryption scheme* EC *is IND-CPA secure. Assume that* Sig *is a signature scheme satisfying EUF-CMA. Then the protocol* $\Pi_{\mathsf{vote}}^{n,m,\ell,t,k}$ *IUC-realizes* $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$ *against static corruption and adaptive coercion in the* $\{\mathcal{F}_{\mathsf{DKG}}^{t,k}[\mathbb{G}], \mathcal{F}_{\mathsf{sc}}, \mathcal{G}_{\mathsf{PBB}}\}$*-hybrid world.*

Proof. To prove the theorem, we construct the real deception strategies DR and a simulator $\mathcal{S}$ such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish (i) the real execution $\mathsf{EXEC}_{\Pi_{\mathsf{vote}}^{n,m,\ell,t,k},\mathsf{DR},\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\mathsf{DKG}}^{t,k}[\mathbb{G}],\mathcal{F}_{\mathsf{sc}},\mathcal{G}_{\mathsf{PBB}}}$ from the (ii) the ideal execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k},\mathsf{DI},\mathcal{S},\mathcal{Z}}^{\mathcal{G}_{\mathsf{PBB}}}$.

Real Deception Strategy. The real deception strategy DR_i internally runs DI_i, forwarding messages between DI_i and the environment $\mathcal{Z}$. DR_i performs as described in Fig. 14.

When the coercer asks for the view of the registration phase, DR_i generates $K' \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_\mathsf{T}}(\mathsf{pk}'_j)$. Then, the voter claims that $(K'_j, A_j, \mathsf{tx}_j, \delta_j)$ is the message sent to the RA by simulating the designated verifier proof of re-encryption.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to and from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the voters $\mathcal{V} := \{\mathsf{V}_1,\ldots,\mathsf{V}_n\}$, the experts $\mathcal{E} := \{\mathsf{E}_1,\ldots,\mathsf{E}_m\}$, the registration authority RA, the shuffler, the trustees $\mathcal{T} := \{\mathsf{T}_1,\ldots,\mathsf{T}_k\}$, and the ideal functionalities $\mathcal{F}_{\mathsf{DKG}}^{t,k}[\mathbb{G}], \mathcal{F}_{\mathsf{sc}}$. It works as follows:

In the preparation phase:

Upon receiving $(\textsc{InitNotify}, \mathsf{sid}, \mathsf{T}_i)$ from the ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$, $\mathcal{S}$ simulates the trustee T_i following the protocol $\Pi_{\mathsf{vote}}^{n,m,\ell,t,k}$ as if he receives $(\textsc{Init}, \mathsf{sid})$ from the environment $\mathcal{Z}$.

Upon receiving $(\textsc{InitNotify}, \mathsf{sid}, \mathsf{RA})$ from the ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$, $\mathcal{S}$ simulates the RA following the protocol $\Pi_{\mathsf{vote}}^{n,m,\ell,t,k}$ as if he receives $(\textsc{Init}, \mathsf{sid})$ from the environment $\mathcal{Z}$.

In the registration phase:

Upon receiving $(\textsc{LeakPower}, \mathsf{sid}, \langle \alpha_i, \mathsf{V}_i \rangle)$ from the ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$, $\mathcal{S}$ simulates the voter V_j following the protocol $\Pi_{\mathsf{vote}}^{n,m,\ell,t,k}$ as if he receives $(\textsc{Reg}, \mathsf{sid}, \alpha_j)$ from the environment $\mathcal{Z}$.

Upon receiving $(\textsc{RegNotify}, \mathsf{sid}, \mathsf{V}_j)$ from the ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$, $\mathcal{S}$ simulates the voter V_j following the protocol $\Pi_{\mathsf{vote}}^{n,m,\ell,t,k}$ as if he receives $(\textsc{Reg}, \mathsf{sid}, 0)$ from the environment $\mathcal{Z}$.

When the the registration authority RA receives $(\textsc{Sent}, \mathsf{sid}, \mathsf{V}_j, \langle K_j, A_j, \mathsf{tx}_j, \delta_j \rangle)$ from the simulated $\mathcal{F}_{\mathsf{sc}}$, $\mathcal{S}$ simulates the RA following the protocol $\Pi_{\mathsf{vote}}^{n,m,\ell,t,k}$.

In the voting/delegation phase:

Upon receiving $(\textsc{VoteNotify}, \mathsf{sid}, \mathsf{VOTER})$ from the ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$, $\mathcal{S}$ simulates an honest voter V_j following the protocol $\Pi_{\mathsf{vote}}^{n,m,\ell,t,k}$ as if he receives $(\textsc{Vote}, \mathsf{sid}, v_0)$ from the environment $\mathcal{Z}$.

Upon receiving $(\textsc{VoteNotify}, \mathsf{sid}, \mathsf{EXPERT})$ from the ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$, $\mathcal{S}$ simulates an honest expert E_i following the protocol $\Pi_{\mathsf{vote}}^{n,m,\ell,t,k}$ as if he receives $(\textsc{Vote}, \mathsf{sid}, v_0)$ from the environment $\mathcal{Z}$.

When a voter V_i is coerced, $\mathcal{S}$ simulates V_i performing the real deceiving strategy DR_i to cast a fake ballot following the coercer's instructions.

Once $\mathcal{G}_{\mathsf{PBB}}$ receives $(\textsc{Write}, \mathsf{sid}, B_i^{\mathsf{E}})$, $\mathcal{S}$ does the following:

- Parse B_i^{E} as $(\mathsf{E}_i, \boldsymbol{c}_i, \tau_i, \sigma_{\mathsf{E}_i})$.
- Check (i) $\mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Verify}(\tau_i, \boldsymbol{c}_i) \overset{?}{=} 1$; (ii) $\mathsf{Sig}.\mathsf{Verify}_{\mathsf{pk}_{\mathsf{E}_i}}(\sigma_{\mathsf{E}_i}, \boldsymbol{c}_i) \overset{?}{=} 1$.
- If it is valid, compute $v_i = \mathsf{EC.Dec}_{\mathsf{sk}_\mathsf{T}}(\boldsymbol{c}_i)$;
- If E_i is not corrupted, $\mathcal{S}$ will abort.
- Send $(\textsc{Vote}, \mathsf{sid}, v_i)$ to $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$ on behalf of E_i.

Once $\mathcal{G}_{\mathsf{PBB}}$ receives $(\textsc{Write}, \mathsf{sid}, B_j)$, $\mathcal{S}$ does the following:

- Parse B_j as $(\mathsf{pk}_j, \boldsymbol{u}_j, \pi_j, \sigma_j)$.
- Check (i) $\mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Verify}(\pi_j, \boldsymbol{u}_j) \overset{?}{=} 1$; (ii) $\mathsf{Sig}.\mathsf{Verify}_{\mathsf{pk}_j}(\sigma_j, \boldsymbol{u}_j) \overset{?}{=} 1$.
- If it is valid, compute $v_j = \mathsf{EC}.\mathsf{Dec}_{\mathsf{sk}_\mathsf{T}}(\boldsymbol{u}_i)$;
- Find the owner of pk_j by decrypting all the registration messages. Denote him as V_j.
- If V_j is not corrupted, $\mathcal{S}$ will abort.
- Send $(\textsc{Vote}, \mathsf{sid}, v_j)$ to $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$ on behalf of V_j.

In the tally phase:

Upon receiving $(\textsc{VoteEnd}, \mathsf{sid})$ from the ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$, $\mathcal{S}$ simulates the shuffler following the protocol $\Pi_{\mathsf{vote}}^{n,m,\ell,t,k}$ as if he receives $(\textsc{VoteEnd}, \mathsf{sid})$ from the environment $\mathcal{Z}$.

Upon receiving $(\textsc{LeakTally}, \mathsf{sid}, \eta, \mathsf{ballots}_\mathsf{E}, \mathsf{bc}_\mathsf{V})$ from the ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$, $\mathcal{S}$ records $\eta, \mathsf{ballots}_\mathsf{E}, \mathsf{bc}_\mathsf{V}$.

Upon receiving $(\textsc{LeakBallots}, \mathsf{sid}, \mathsf{ballots})$ from the ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$, $\mathcal{S}$ records $\mathsf{ballots}$.

Upon receiving $(\textsc{TallyNotify}, \mathsf{sid}, \mathsf{T}_i)$ from the ideal functionality $\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k}$, $\mathcal{S}$ does the following:

- Set $\mathcal{J} := \mathcal{J} \cup \{\mathsf{T}_i\}$.
- If $|\mathcal{J} \cap \mathcal{T}_{\mathsf{honest}}| + |\mathcal{T}_{\mathsf{cor}}| < t$, $\mathcal{S}$ simulates the trustee T_i following the protocol $\Pi_{\mathsf{vote}}^{n,m,\ell,t,k}$ as if he receives $(\textsc{Tally}, \mathsf{sid})$ from the environment $\mathcal{Z}$.
- Otherwise, $\mathcal{S}$ simulates T_i's decryption and the corresponding NIZK based on the tally result η and the known information about ballots.

Indistinguishability.

We prove indistinguishability through a series of hybrid worlds $\mathcal{H}_0, \ldots, \mathcal{H}_6$.

Hybrid $\mathcal{H}_0$: This is the real world execution $\mathsf{EXEC}_{\Pi_{\mathsf{vote}}^{n,m,\ell,t,k},\mathsf{DR},\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\mathsf{DKG}}^{t,k}[\mathbb{G}],\mathcal{F}_{\mathsf{sc}},\mathcal{G}_{\mathsf{PBB}}}$.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that during the tally phase, the honest trustees' decryption NIZKs are generated by the NIZK simulator.

<u>Claim 1:</u> If $\mathsf{NIZK}_{\mathsf{Dec}}$ is zero-knowledge with adversary advantage $\mathsf{Adv}_{\mathsf{NIZK}_{\mathsf{Dec}}}^{\mathsf{zk}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_1$ and $\mathcal{H}_0$ are indistinguishable with distinguishing advantage at most $(4n + m + \ell) \cdot \mathsf{Adv}_{\mathsf{NIZK}_{\mathsf{Dec}}}^{\mathsf{zk}}(\mathcal{A}, \lambda)$.

<u>Proof 1:</u> With each voting casting at most one fake ballot in our modeling, there are at most $2n$ voters' ballots and $2n$ encrypted public key items. For experts and candidates, each of them has one ciphertext to decrypt. Therefore, the overall advantage is at most $(4n + m + \ell) \cdot \mathsf{Adv}_{\mathsf{NIZK}_{\mathsf{Dec}}}^{\mathsf{zk}}(\mathcal{A}, \lambda)$ by a standard hybrid argument.

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except that during the tally phase, the honest trustees' decryption shares are backward calculated from the tally result.

<u>Claim 2:</u> $\mathcal{H}_2$ and $\mathcal{H}_1$ are perfectly indistinguishable.

<u>Proof 2:</u> In out threshold cryptosystem, the backward calculated shares in $\mathcal{H}_2$ and the shares in $\mathcal{H}_1$ have the same distribution.

Hybrid $\mathcal{H}_3$: $\mathcal{H}_3$ is the same as $\mathcal{H}_2$ except that in the voting/delegation phase, the honest voters' ballots are replaced with ballots for candidate v_0.

<u>Claim 3:</u> If the encryption scheme EC is IND-CPA with advantage $\mathsf{Adv}_{\mathsf{EC}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A}, \lambda)$ and $\mathsf{NIZK}_{\mathsf{knowledge}}$ is zero-knowledge with adversary advantage $\mathsf{Adv}_{\mathsf{NIZK}_{\mathsf{knowledge}}}^{\mathsf{zk}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_3$ and $\mathcal{H}_2$ are indistinguishable with distinguishing advantage at most $n \cdot \mathsf{Adv}_{\mathsf{EC}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A}, \lambda) + n \cdot \mathsf{Adv}_{\mathsf{NIZK}_{\mathsf{knowledge}}}^{\mathsf{zk}}(\mathcal{A}, \lambda)$.

<u>Proof 3:</u> With at most n honest voters in the system, the overall advantage is at most $n \cdot \mathsf{Adv}_{\mathsf{EC}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A}, \lambda) + n \cdot \mathsf{Adv}_{\mathsf{NIZK}_{\mathsf{knowledge}}}^{\mathsf{zk}}(\mathcal{A}, \lambda)$ by a standard hybrid argument.

Hybrid $\mathcal{H}_4$: $\mathcal{H}_4$ is the same as $\mathcal{H}_3$ except that if a corrupted voter generates a valid ballot for an honest voter, the execution will abort.

<u>Claim 4:</u> If the signature scheme Sig is EUF-CMA with advantage $\mathsf{Adv}_{\mathsf{Sig}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_4$ and $\mathcal{H}_3$ are indistinguishable with distinguishing advantage at most $n \cdot \mathsf{Adv}_{\mathsf{Sig}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}, \lambda)$.

<u>Proof 4:</u> There are at most n honest voters, so the probability of abortion is no more than $n \cdot \mathsf{Adv}_{\mathsf{Sig}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}, \lambda)$ by a standard hybrid argument.

Hybrid $\mathcal{H}_5$: $\mathcal{H}_5$ is the same as $\mathcal{H}_4$ except that if a corrupted expert generates a valid ballot for an honest expert, the execution will abort.

<u>Claim 5:</u> If the signature scheme Sig is EUF-CMA with advantage $\mathsf{Adv}_{\mathsf{Sig}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_5$ and $\mathcal{H}_4$ are indistinguishable with distinguishing advantage at most $m \cdot \mathsf{Adv}_{\mathsf{Sig}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}, \lambda)$.

<u>Proof 5:</u> Same as the previous proof, the probability of abortion is no more than $m \cdot \mathsf{Adv}_{\mathsf{Sig}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}, \lambda)$ by a standard hybrid argument.

Hybrid $\mathcal{H}_6$: This is the ideal execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{vote}}^{n,m,\ell,t,k},\mathsf{DI},\mathcal{S},\mathcal{Z}}^{\mathcal{G}_{\mathsf{PBB}}}$.

<u>Claim 6:</u> If the decryption NIZK $\mathsf{NIZK}_{\mathsf{Dec}}$ is sound with adversary advantage $\mathsf{Adv}_{\mathsf{NIZK}_{\mathsf{Dec}}}^{\mathsf{sound}}(\mathcal{A}, \lambda)$ and the shuffle NIZK $\mathsf{NIZK}_{\mathsf{shuffle}}$ is sound with adversary advantage $\mathsf{Adv}_{\mathsf{NIZK}_{\mathsf{shuffle}}}^{\mathsf{sound}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_6$ and $\mathcal{H}_5$ are indistinguishable with distinguishing advantage at most $(4n + m + \ell) \cdot \mathsf{Adv}_{\mathsf{NIZK}_{\mathsf{Dec}}}^{\mathsf{sound}}(\mathcal{A}, \lambda) + \mathsf{Adv}_{\mathsf{NIZK}_{\mathsf{shuffle}}}^{\mathsf{sound}}(\mathcal{A}, \lambda)$.

<u>Proof 6:</u> It suffices to argue that the ideal tally and the real tally output the same result. We can see that, as long as the re-encryption in the registration phase is correct and the shuffle and decryption in the tally phase are sound, the ideal tally and the real tally did the same computation by the additive homomorphism of lifted ElGamal encryption scheme. Thus, the overall advantage is no more than $n \cdot \mathsf{Adv}_{\mathsf{NIZK}_{\mathsf{DVP\text{-}reenc}}}^{\mathsf{sound}}(\mathcal{A}, \lambda) +$

$(4n+m+\ell)\cdot\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{NIZK}_{\mathsf{Dec}}}(\mathcal{A},\lambda)+\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{NIZK}_{\mathsf{shuffle}}}(\mathcal{A},\lambda)$.

Combining together, the real execution $\mathsf{EXEC}^{\mathcal{F}^{t,k}_{\mathsf{DKG}}[\mathbb{G}],\mathcal{F}_{\mathsf{sc}},\mathcal{G}_{\mathsf{PBB}}}_{\Pi^{n,m,\ell,t,k}_{\mathsf{vote}},\mathsf{DR},\mathcal{A},\mathcal{Z}}$ and the ideal execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{PBB}}}_{\mathcal{F}^{n,m,\ell,t,k}_{\mathsf{vote}},\mathsf{DI},\mathcal{S},\mathcal{Z}}$ are indistinguishable with distinguishing advantage at most

$$\begin{aligned}&(4n+m+\ell)\cdot\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{Dec}}}(\mathcal{A},\lambda)+n\cdot\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{EC}}(\mathcal{A},\lambda)\\&+n\cdot\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{knowledge}}}(\mathcal{A},\lambda)+(n+m)\cdot\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig}}(\mathcal{A},\lambda)\\&+n\cdot\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{NIZK}_{\mathsf{DVP\text{-}reenc}}}(\mathcal{A},\lambda)+(4n+m+\ell)\cdot\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{NIZK}_{\mathsf{Dec}}}(\mathcal{A},\lambda)\\&+\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{NIZK}_{\mathsf{shuffle}}}(\mathcal{A},\lambda)\end{aligned}$$

This concludes the proof.

□