A Scalable Coercion-resistant Voting Scheme for Blockchain Decision-making

Zeyuan Yin
The State Key Laboratory of Blockchain and Data Security, Zhejiang University
zeyuanyin@zju.edu.cn

Bingsheng Zhang
The State Key Laboratory of Blockchain and Data Security, 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
The State Key Laboratory of Blockchain and Data Security, 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 voting 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. We formally prove that our scheme has ballot privacy, verifiability, and coercion-resistance. We implement a prototype of the scheme and the evaluation result shows that our scheme's tally procedure is more than 6x faster than VoteAgain (USENIX '20) in an election with over 10,000 voters and over 50% extra ballot rate.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; *Cryptography*; *Pseudonymity, anonymity and untraceability*; *Distributed systems security.*

KEYWORDS

Electronic voting, coercion-resistant, blockchain decision-making, remote voting, liquid democracy

1 INTRODUCTION

Blockchain technology enjoys its popularity since the invention of Bitcoin in 2008, and it continues to reshape our digital society with its properties of 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 blockchain voting, each participant's voting power is different and 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, stakeholders must invest substantial effort into acquiring knowledge and expertise throughout the decision-making process. Therefore, letting elites lead the decision-making might be an optimization in most cases. Some systems such as ZCash [8] 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 [26]) 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.* [45] 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 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 [43], Helios [2], and prêt à voter [42]. 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, 19, 34], re-voting [1, 30, 39, 40], and secure hardware [3, 41]. 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 the fact 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-voting* 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, 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 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 [34] 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 [40], 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 [34]. 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 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 match", 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 formally prove that our scheme meets the game-based definitions of ballot privacy, verifiability, and coercion-resistance. We implement the scheme and evaluate its performance. Results show that our scheme's tally execution time is more than 6x faster than VoteAgain [40] in elections with over 10,000 voters and over 50% extra ballot rate[1].

1.2 Related Work

Coercion-resistant voting can be roughly split into three classes: fake credentials [4, 15, 17, 19, 20, 34], re-voting [1, 30, 39, 40], and secure hardware [3, 41]. JCJ [34] is the first paper that introduces the *"fake credentials"* type of coercion-resistant voting. In JCJ, each

[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.

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 a 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.

Schemes	Diff. voting power	Del.	Ballot size	Complexity	Crypto state	Ballot privacy	Verifiability	Coercion-resistant
JCJ [34] (fake credentials)	No	No	$O(1)$	$O(n^2)$	Yes	t-out-of-k	trust no one	trust EA
ABBT [4] (fake credentials)	No	No	$O(1)$	$O(n)$	Yes	t-out-of-k	trust no one	trust EA
AKL+ [1], LHK [39] (re-voting)	No	No	$O(1)$	$O(n^2)$	No	t-out-of-k	trust no one	secret credential
VoteAgain [40] (re-voting)	No	No	$O(1)$	$O(n \log n)$	No	t-out-of-k	trust EA	trust EA
MBC [41], AOZZ [3] (secure hardware)	No	No	$O(1)$	$O(n)$	No	hardware	hardware	hardware
Snapshot [43]	Yes	No	$O(1)$	$O(n)$	-	t-out-of-k	trust no one	-
ZOB [45]	Yes	Yes	$O(m)$	$O(mn)$	-	t-out-of-k	trust no one	-
Our scheme (fake credentials)	**Yes**	**Yes**	$\boldsymbol{O(1)}$	$\boldsymbol{O(n)}$	Yes	$\boldsymbol{t}$**-out-of-**$\boldsymbol{k}$	**trust no one**	trust EA

ballot contains an encrypted credential, and there is a list of encrypted valid credentials on 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 cause $O(n^2)$ complexity, where n is the number of voters. Other "fake credentials" schemes such as [4, 19] 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.* [39] utilize a deniable vote update mechanism to realize re-voting with quadratic complexity. The Norwegian Internet voting protocol [30] and VoteAgain [40] achieve (quasi-)linear complexity, but they both need a trusted third party for verifiability. Schemes based on *secure hardware* [3, 41] can achieve coercion-resistance easily, but secure hardware is a strong assumption. A recent work by Giustolisi *et al.* [29] proposes the first re-voting scheme that can evade last-minute voter coercion. The scheme has a re-voting form, but it requires the voter to remember a secret (number of cast ballots), so it has similar assumptions as "fake credentials" schemes. The scheme can be modified to support differential voting power; however, in the scheme, the voting server needs to continuously generate obfuscation ballots for each voter, so its complexity is linear to the duration of voting phase and does not achieve $O(n)$ complexity.

On the other hand, blockchain voting [38, 43–45] is becoming more and more popular nowadays. Snapshot [43] is a popular DAO (Decentralized Autonomous Organization) voting platform that frees voters from gas fees. It uses IPFS [9] to store the proposals and votes, making the voting process off-chain and gas-free. Zhang *et al.* [45] 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 SYSTEM OVERVIEW

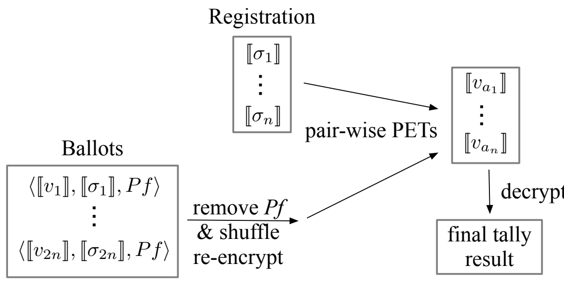
In this section, we start by modifying the JCJ protocol [34] to build a coercion-resistant voting scheme that supports differential voting power with $O(n^2)$ complexity. Then, we give the intuition and details of our novel "dummy voting power" technique and "two-layer tally" procedure. We also optimize JCJ's ballot structure to achieve a smaller ballot size. Finally, we provide an overview of our scheme.

2.1 The JCJ Protocol

We first recall the well-known JCJ protocol [34] and show how to modify it to support differential voting power.

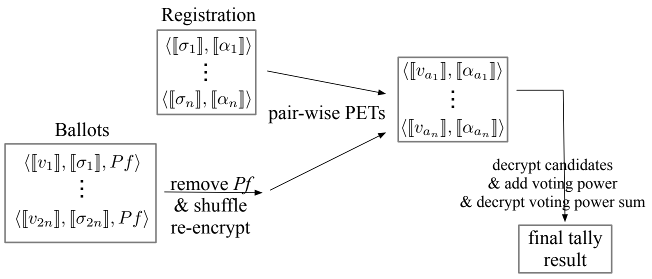
The original protocol. As mentioned above, JCJ is the first protocol that introduces the concept of "fake credentials". Generally, it works as follows. For simplicity, we use $[\![x]\!]$ to denote encryption of x in section 2.1 and 2.2. In the registration phase, a voter authenticates to the election authority (EA), and the EA generates a credential $\sigma \leftarrow \mathbb{G}$. Then, the EA publishes $S = [\![\sigma]\!]$ on the BB (bulletin board) 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, $[\![\sigma]\!]$ is the encryption of a credential σ, 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 Figure 1.

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 BB, where $[\![\sigma]\!]$ is the encrypted credential and $[\![\alpha]\!]$ is the encrypted voting power under an additively 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,



Registration
$[\![\sigma_1]\!]$ ⋮ $[\![\sigma_n]\!]$
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 1: JCJ original protocol



Registration
$\langle [\![\sigma_1]\!], [\![\alpha_1]\!] \rangle$ ⋮ $\langle [\![\sigma_n]\!], [\![\alpha_n]\!] \rangle$
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 2: JCJ protocol with differential voting power

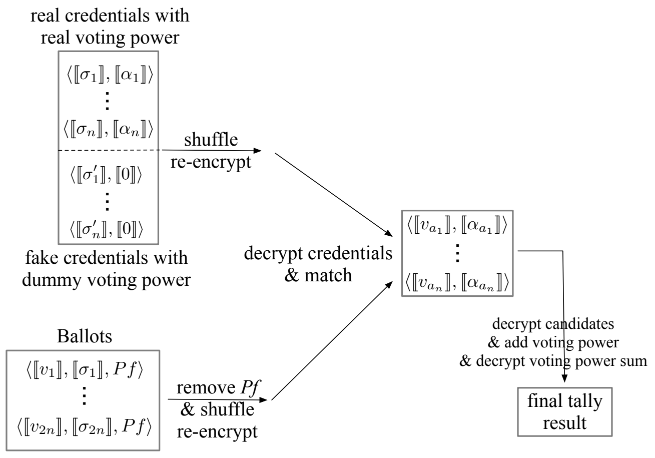
the trustees decrypt the candidates and add the voting power by additive homomorphism. The overview of the modified JCJ protocol with differential voting power is shown in Figure 2.

2.2 Our Technique

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

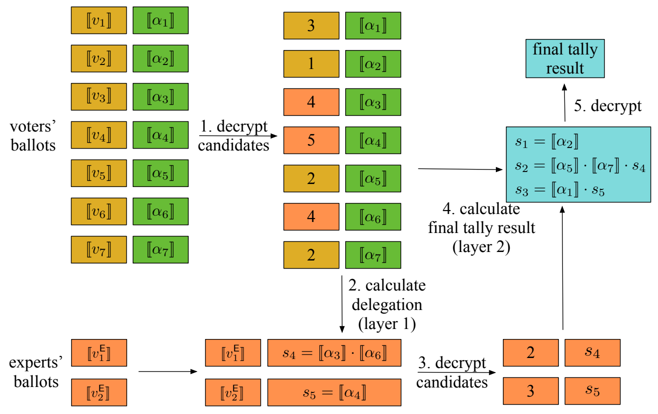
Dummy voting power. We can see that in the JCJ protocol, the pair-wise PETs cause $O(n^2)$ complexity. The idea is that if we allow voters to publish the fake credentials on the BB but associate them with dummy (zero) voting power, then we can directly decrypt the credentials instead of performing PETs 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 PETs 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 Figure 3.

Two-layer tally. To support delegation, the trustees perform a "two-layer tally" procedure 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. Figure 4 shows the process of "two-layer tally" where there are 3 candidates and 2 experts.



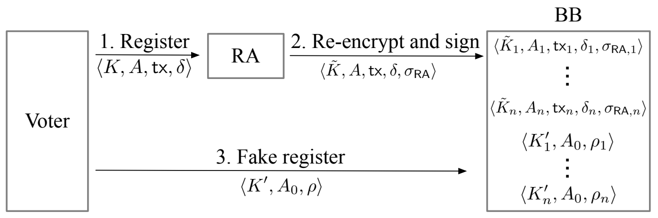
real credentials with real voting power
$\langle [\![\sigma_1]\!], [\![\alpha_1]\!] \rangle$ ⋮ $\langle [\![\sigma_n]\!], [\![\alpha_n]\!] \rangle$
$\langle [\![\sigma'_1]\!], [\![0]\!] \rangle$ ⋮ $\langle [\![\sigma'_n]\!], [\![0]\!] \rangle$
fake credentials with dummy voting power
shuffle re-encrypt
decrypt credentials & match
$\langle [\![v_{a_1}]\!], [\![\alpha_{a_1}]\!] \rangle$ ⋮ $\langle [\![v_{a_n}]\!], [\![\alpha_{a_n}]\!] \rangle$
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 3: Dummy voting power technique



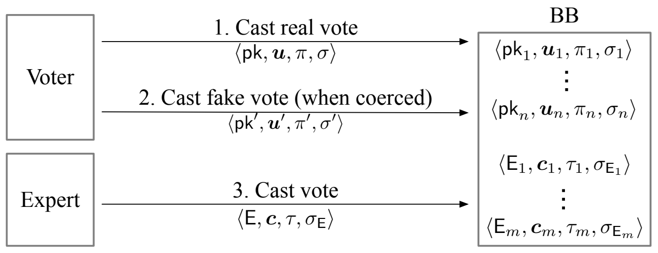
voters' ballots
$[\![v_1]\!]$ $[\![\alpha_1]\!]$
$[\![v_2]\!]$ $[\![\alpha_2]\!]$
$[\![v_3]\!]$ $[\![\alpha_3]\!]$
$[\![v_4]\!]$ $[\![\alpha_4]\!]$
$[\![v_5]\!]$ $[\![\alpha_5]\!]$
$[\![v_6]\!]$ $[\![\alpha_6]\!]$
$[\![v_7]\!]$ $[\![\alpha_7]\!]$
1. decrypt candidates
3 $[\![\alpha_1]\!]$
1 $[\![\alpha_2]\!]$
4 $[\![\alpha_3]\!]$
5 $[\![\alpha_4]\!]$
2 $[\![\alpha_5]\!]$
4 $[\![\alpha_6]\!]$
2 $[\![\alpha_7]\!]$
final tally result
5. decrypt
$s_1 = [\![\alpha_2]\!]$
$s_2 = [\![\alpha_5]\!] \cdot [\![\alpha_7]\!] \cdot s_4$
$s_3 = [\![\alpha_1]\!] \cdot s_5$
4. calculate final tally result (layer 2)
2. calculate delegation (layer 1)
experts' ballots
$[\![v^{\mathsf{E}}_1]\!]$
$[\![v^{\mathsf{E}}_2]\!]$
$[\![v^{\mathsf{E}}_1]\!]$ $s_4 = [\![\alpha_3]\!] \cdot [\![\alpha_6]\!]$
$[\![v^{\mathsf{E}}_2]\!]$ $s_5 = [\![\alpha_4]\!]$
3. decrypt candidates
2 s_4
3 s_5

Figure 4: Two-layer tally



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

Figure 5: Registration phase



BB
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 6: Voting/delegation phase

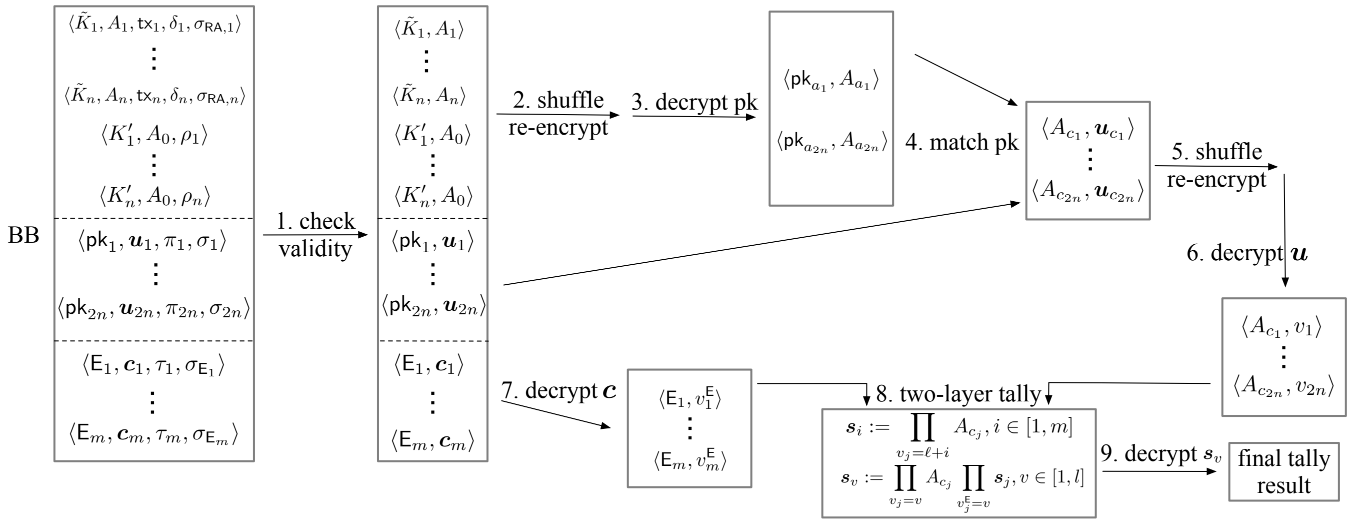

BB
$\langle \tilde{K}_1, A_1, \mathsf{tx}_1, \delta_1, \sigma_{\mathsf{RA},1}\rangle$
$\langle \tilde{K}_n, A_n, \mathsf{tx}_n, \delta_n, \sigma_{\mathsf{RA},n}\rangle$
$\langle K'_1, A_0, \rho_1\rangle$
$\langle K'_n, A_0, \rho_n\rangle$
$\langle \mathsf{pk}_1, \boldsymbol{u}_1, \pi_1, \sigma_1\rangle$
$\langle \mathsf{pk}_{2n}, \boldsymbol{u}_{2n}, \pi_{2n}, \sigma_{2n}\rangle$
$\langle \mathsf{E}_1, \boldsymbol{c}_1, \tau_1, \sigma_{\mathsf{E}_1}\rangle$
$\langle \mathsf{E}_m, \boldsymbol{c}_m, \tau_m, \sigma_{\mathsf{E}_m}\rangle$
1. check validity
$\langle \tilde{K}_1, A_1\rangle$
$\langle \tilde{K}_n, A_n\rangle$
$\langle K'_1, A_0\rangle$
$\langle K'_n, A_0\rangle$
$\langle \mathsf{pk}_1, \boldsymbol{u}_1\rangle$
$\langle \mathsf{pk}_{2n}, \boldsymbol{u}_{2n}\rangle$
$\langle \mathsf{E}_1, \boldsymbol{c}_1\rangle$
$\langle \mathsf{E}_m, \boldsymbol{c}_m\rangle$
2. shuffle re-encrypt
3. decrypt pk
$\langle \mathsf{pk}_{a_1}, A_{a_1}\rangle$
$\langle \mathsf{pk}_{a_{2n}}, A_{a_{2n}}\rangle$
4. match pk
$\langle A_{c_1}, \boldsymbol{u}_{c_1}\rangle$
$\langle A_{c_{2n}}, \boldsymbol{u}_{c_{2n}}\rangle$
5. shuffle re-encrypt
6. decrypt $\boldsymbol{u}$
$\langle A_{c_1}, v_1\rangle$
$\langle A_{c_{2n}}, v_{2n}\rangle$
7. decrypt $\boldsymbol{c}$
$\langle \mathsf{E}_1, v_1^{\mathsf{E}}\rangle$
$\langle \mathsf{E}_m, v_m^{\mathsf{E}}\rangle$
8. two-layer tally
$\boldsymbol{s}_i := \prod_{v_j=\ell+i} A_{c_j}, i \in [1,m]$
$\boldsymbol{s}_v := \prod_{v_j=v} A_{c_j} \prod_{v_j^{\mathsf{E}}=v} \boldsymbol{s}_j, v \in [1,l]$
9. decrypt $\boldsymbol{s}_v$
final tally result

Figure 7: Tally phase

2.3 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.
- *The shuffler* performs verifiable shuffle procedures on the ciphertexts.
- *The trustees* are responsible for decrypting the ballot and revealing the final tally result.

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

An optimization of JCJ's ballot structure. We further 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 of them will be decrypted in the tally phase. If it encrypts an invalid value, we can simply treat it as "abstain" and drop it. Secondly, instead of defining σ as the secret credential, we can define the discrete logarithm of σ as the secret credential (voting secret key) and define σ as the voting public key, i.e., $\sigma := \mathsf{pk} := g^{\mathsf{sk}}$. Then, the ballot can be modified as $\langle \mathsf{pk}, \boldsymbol{u}, \pi, Sig\rangle$, where $\boldsymbol{u} := [\![v]\!]$ is the encrypted choice, π is a NIZK proof of plaintext knowledge of $\boldsymbol{u}$, and $Sig \leftarrow \mathsf{Sign}(\mathsf{sk}, \boldsymbol{u})$. By doing so, we change an Elgamal ciphertext $[\![\sigma]\!]$ to a group element pk, achieving a 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 they share sk_{T}.

In the registration phase (see Figure 5), each voter first freezes some stake by transaction tx. Then, he generates a pair of real voting keys $\langle \mathsf{pk}, \mathsf{sk}\rangle$ and sends a registration message $\langle K, A, \mathsf{tx}, \delta\rangle$ to the RA (step 1), where $K \leftarrow \mathsf{EC.Enc}_{\mathsf{pk_T}}(\mathsf{pk})$ is encryption of real public key, $A \leftarrow \mathsf{LE.Enc}_{\mathsf{pk_T}}(\alpha)$ is encryption of voting power, tx is the transaction that freezes some stake, and δ is the NIZK proof that the encrypted voting power equals the frozen stake (Cf. Appendix B). After authenticating the voter (i.e., checking that the voter knows the sk corresponding to tx's sender's pk), 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 the "registration item" $\langle \tilde{K}, A, \mathsf{tx}, \delta, \sigma_{\mathsf{RA}}\rangle$ to the BB (step 2), where $\sigma_{\mathsf{RA}} \leftarrow \mathsf{Sig.sign}_{\mathsf{sk_{RA}}}(\tilde{K}||A||\mathsf{tx}||\delta)$. At any convenient time, the voter can generate a pair of fake voting keys $\langle \mathsf{pk}', \mathsf{sk}'\rangle$ and publish a "fake registration item" $\langle K', A_0, \rho\rangle$ on the BB (step 3), where $K' \leftarrow \mathsf{EC.Enc}_{\mathsf{pk_T}}(\mathsf{pk}')$ is encryption of fake public key, A_0 is a deterministic encryption of 0, and ρ is a NIZK proof of knowledge of sk'. The voter can repeat step 3 multiple times to generate multiple fake keys. In this phase, all the users who want to be experts will also register by authenticating to the blockchain (by signing a registration message using the blockchain secret key).

In the voting phase (see Figure 6), all registered experts $\mathsf{E}_1, \ldots, \mathsf{E}_m$ form a list of eligible experts. Each voter encrypts his choice with the trustees' public key $\mathsf{pk_T}$, signs it with the voting secret key, and casts it on BB (step 1). Specifically, a voter's ballot is of the form $\langle \mathsf{pk}, \boldsymbol{u}, \pi, \sigma\rangle$, where $\boldsymbol{u} \leftarrow \mathsf{EC.Enc}_{\mathsf{pk_T}}(v)$ is the encrypted choice, π is a NIZK proof of plaintext knowledge of $\boldsymbol{u}$, and $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \boldsymbol{u})$ is the signature. 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 $\mathsf{EC.Enc}_{\mathsf{pk_T}}(\mathsf{pk}')$ 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, $\boldsymbol{c} \leftarrow \mathsf{EC.Enc}_{\mathsf{pk_T}}(v^{\mathsf{E}})$ is the encrypted candidate, τ is the NIZK proof of plaintext knowledge, and σ_{E} is the signature.

In the tally phase (see Figure 7), the BB contains (real and fake) "registration items", voters' ballots, and experts' ballots at the beginning. For simplicity, in Fig. 7, 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 "registration items" and ballots, and removes the NIZK proofs and signatures (step 1). Then, the shuffler shuffle re-encrypts the "registration items" (step 2). Next, the trustees jointly decrypt the public keys in "registration items" and voters' ballots (step 3). If the same public key appears more than once, drop them. The encrypted voting power and encrypted choices with the same public key will be matched (step 4). To ensure ballot privacy, a shuffle re-encryption is performed on the matched items (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, each expert's 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 properly. Also, there should be a validator to check 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 (see sec. 3 for details). Therefore, it may be instantiated with a trusted execution environment (TEE) like Intel SGX.
The shuffler. The shuffler can be implemented by a mixnet [16], and the mixnet nodes can be selected by cryptographic sortition [18]. Ballot privacy is preserved as long as one mixnet node is honest.
The trustees. The trustees can also be selected by cryptographic sortition [18]. In the blockchain context, the majority of trustees are honest with a high probability when the 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 are 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 BB. It also plays the role of PKI in the registration phase to authenticate voters and experts.

3 THREAT MODEL

We consider an adversary whose goal is to coerce voters into casting ballots for a particular candidate or to abstain. We rely on three assumptions: 1) anonymous communication with honest BB, 2) untappable channels, and 3) inalienable authentication. While these assumptions are strong, we will argue that they are necessary assumptions for all "fake credentials" schemes and explain how they can be achieved in the blockchain context.

Assumption 1. The bulletin board is honest, and the communication with the BB is anonymous.

BB is a basic assumption for any electronic voting scheme. A malicious BB can break verifiability by creating different views for different voters [32]. Then, ballot privacy will be undermined if ballots can be dropped undetectably [22], and it is not possible to achieve coercion-resistance without ballot privacy. Thus, BB is trusted for all three properties. Moreover, communication with BB 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 BB. A voter can use anonymous channels (e.g., TOR) to broadcast on the blockchain. Although there are attacks that leverage the designs or implementation of the blockchain to deanonymize a user [13, 37], in this work, we assume that the communication with the blockchain can be anonymous, as in [27, 36]. How to achieve anonymous communication with the blockchain in practice is out of the scope of this paper.

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 [33].

As mentioned above, the RA can be instantiated with TEE, such as Intel SGX. A voter can communicate with the RA using TOR.

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 will informally define ballot privacy, verifiability, and coercion-resistance and analyze how they are achieved. In section 5, we will formalize these properties and provide a security proof.
Definition 1. (Ballot privacy) The adversary cannot learn the votes of honest voters.
Definition 2. (Verifiability) Honest voters' ballots must be tallied, and the adversary cannot cast more votes than the number of voters that he controls.
Definition 3. (Coercion-resistance) A coercer cannot determine if the coerced party is trying to deceive him.
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

Table 2: Trust assumptions on the entities.

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

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 registration items will be shuffled before decryption so that the link between identity and ballot/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 [28]. In the registration phase, we use two NIZKs to ensure that (i) the encrypted voting power is equal to the frozen stake and (ii) the RA does the re-encryption correctly. In the voting/delegation phase, the 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 [6], and decryption correctness is guaranteed by the decryption NIZK [28]. 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 registration 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 [16] can be utilized, and we can perform a cryptographic sortition [18] 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.

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 an 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 a TEE to instantiate the RA on the blockchain.

4 THE PROTOCOL

In this section, we will give a detailed protocol description of our voting scheme.

4.1 Cryptographic primitives

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 denote the integers modulo p with $\mathbb{Z}_p$ and write $r \leftarrow_\$ \mathbb{Z}_p$ for r being chosen uniformly from $\mathbb{Z}_p$. We abbreviate *probabilistic polynomial time* as PPT.

(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 $\mathsf{EC.Enc}_{\mathsf{pk}}(m; r)$ takes as input the public key pk, the message $m \in \mathbb{G}$, and randomness $r \leftarrow_\$ \mathbb{Z}_p$, and it 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 $\mathsf{EC.Rand}_{\mathsf{pk}}(c; r)$ takes as input a ciphertext $c := (c_1, c_2)$ and randomenss $r \leftarrow_\$ \mathbb{Z}_p$, and it 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 $\mathsf{LE.Enc}_{\mathsf{pk}}(m; r)$ takes as input the public key pk, the message m, and randomness $r \leftarrow_\$ \mathbb{Z}_p$, and it 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).

The (lifted) ElGamal encryption scheme is IND-CPA secure under the DDH assumption (see Appendix A for formal definition). We omit the randomness r when it is not crucial to the context. Lifted ElGamal encryption is additively 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, with distributed key generation [28], (lifted) ElGamal encryption can be distributed as a threshold encryption scheme.

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. The existential unforgeability under chosen message attack (EUF-CMA) property of a signature scheme is formally defined in Appendix A.

Non-interactive Zero-knowledge Proof (NIZK). Our scheme utilizes Sigma protocols and uses Fiat-Shamir heuristic [25] to transform them into non-interactive proofs of knowledge. There are six NIZK protocols in our scheme for proving: (i) voting power correctness ($\mathsf{NIZK}_{\mathsf{power}}$); (ii) ElGamal encryption plaintext knowledge

($\mathsf{NIZK}_{\mathsf{knowledge}}$); (iii) re-encryption correctness ($\mathsf{NIZK}_{\mathsf{DVP\text{-}reenc}}$); (iv) knowledge of secret key ($\mathsf{NIZK}_{\mathsf{sk}}$); (v) shuffle correctness ($\mathsf{NIZK}_{\mathsf{shuffle}}$); and (vi) decryption correctness ($\mathsf{NIZK}_{\mathsf{Dec}}$). For simplicity, we sometimes omit the witness used in NIZK.Prove() when it is clear in context. We define completeness, simulation-sound extractability, and zero-knowledge of a NIZK in Appendix A, and we give the constructions of these NIZK protocols in Appendix B.

Distributed Key Generation In our scheme, the trustees will run a distributed key generation protocol $\mathsf{Vote.DKeyGen}(\mathbb{G}, t, k)$ for threshold key generation, where $\mathbb{G}$ is the group, t is the corruption threshold, and k is the number of trustees. The protocol outputs a public key pk, and each trustee T_i obtains a share of the secret key. We use the protocol by Gennaro *et al.* [28] to realize the threshold distributed key generation.

4.2 Protocol Description

Preparation Phase:

In the preparation phase, the trustees and the RA prepare their cryptographic materials. Formally:

Procedure 1 (Setup). To setup an election with ℓ candidates $C := \{\mathsf{C}_1, \ldots, \mathsf{C}_\ell\}$, k trustees, and threshold t. The parties proceed as follows:

(1) The trustees run $\mathsf{Vote.DKeyGen}(\mathbb{G}, t, k)$ to generate a public encryption key pk_T and each trustee T_i obtains $\mathsf{sk}_{\mathsf{T},i}$, a share of the decryption key. They publish pk_T.
(2) The RA runs $(\mathsf{pk}_{\mathsf{RA}}, \mathsf{sk}_{\mathsf{RA}}) \leftarrow \mathsf{Sig.Keygen}(1^\lambda)$. It publishes $\mathsf{pk}_{\mathsf{RA}}$.

Registration Phase:

In the registration phase, a voter authenticates to the RA to publish a "registration item". At any convenient time, it can publish "fake registration items" on the BB. Experts also need to register in this phase.

Procedure 2 (Reg(Auth)). A voter takes as input his blockchain inalienable authentication method Auth (usually by proving knowledge of secret key) and does the following:

(1) Assume that the voter issued a privacy-preserving transaction tx that locks α stake during the election.
(2) The voter runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sig.Keygen}(1^\lambda)$.
(3) The voter computes $K \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_\mathsf{T}}(\mathsf{pk})$ and $A \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_\mathsf{T}}(\alpha)$.
(4) The voter generates $\delta \leftarrow \mathsf{NIZK}_{\mathsf{power}}.\mathsf{Prove}(\mathsf{tx}, A)$ proving that A is an encryption of the amount of stake locked in tx.
(5) The voter sends $\langle K, A, \mathsf{tx}, \delta\rangle$ to the RA and uses Auth to authenticate to the RA. The RA checks $\mathsf{NIZK.verify}(A, \mathsf{tx}, \delta) \overset{?}{=} 1$. If the check passes, the RA computes $\tilde{K} \leftarrow \mathsf{EC.Rand}_{\mathsf{pk}_\mathsf{T}}(K)$ and generates the signature $\sigma_{\mathsf{RA}} \leftarrow \mathsf{Sig.sign}_{\mathsf{sk}_{\mathsf{RA}}}(\tilde{K}||A||\mathsf{tx}||\delta)$. Then, the RA publishes the registration item $\langle \tilde{K}, A, \mathsf{tx}, \delta, \sigma_{\mathsf{RA}}\rangle$ on the BB.
(6) The RA generates $\pi_{\mathsf{DVP}} \leftarrow \mathsf{NIZK}_{\mathsf{DVP\text{-}reenc}}.\mathsf{Prove}(\tilde{K}, K)$ and sends π_{DVP} to the voter, where π_{DVP} is a designated verifier proof of re-encryption correctness.
(7) The voter checks $\mathsf{NIZK}_{\mathsf{DVP\text{-}reenc}}.\mathsf{Verify}(\tilde{K}, K, \pi_{\mathsf{DVP}}) \overset{?}{=} 1$. If the check does not pass, he raises a complaint.

Procedure 3 (FakeReg()). At any convenient time, a voter can register a fake key with zero voting power.

(1) The voter runs $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Sig.Keygen}(1^\lambda)$.
(2) The voter computes $K' \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_\mathsf{T}}(\mathsf{pk}')$ and $A_0 = \mathsf{LE.Enc}_{\mathsf{pk}_\mathsf{T}}(0; 1)$.
(3) The voter generates $\rho \leftarrow \mathsf{NIZK}_{\mathsf{sk}}.\mathsf{Prove}(K')$.
(4) The voter publishes the fake registration item $\langle K', A_0, \rho\rangle$ on the BB.

Procedure 4 (E_Reg(Auth)). In the registration phase, all users who want to be experts also register using the blockchain authentication method Auth.

(1) The user i uses Auth to generate an authencation message e_i and publishes $\langle \mathsf{E}_i, e_i\rangle$ on the BB.

Voting/Delegation Phase:

Procedure 5 ($\mathsf{V\_Vote}(\mathsf{pk}, \mathsf{sk}, v)$). To cast a vote, a voter takes as input his key-pair $\langle \mathsf{pk}, \mathsf{sk}\rangle$ and his choice v, and proceeds as follows:

(1) The voter reads the list of eligible experts from the BB, denoted as $\mathsf{E} := \{\mathsf{E}_1, \ldots, \mathsf{E}_m\}$.
(2) The voter computes $\boldsymbol{u} \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_\mathsf{T}}(v)$ and generates $\pi \leftarrow \mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Prove}(\boldsymbol{u})$, which is a proof of plaintext knowledge of $\boldsymbol{u}$.
(3) The voter computes $\sigma \leftarrow \mathsf{Sig.sign}_{\mathsf{sk}}(\boldsymbol{u})$.
(4) The voter sends the ballot $\beta_\mathsf{V} = (\mathsf{pk}, \boldsymbol{u}, \pi, \sigma)$ to the BB.

Procedure 6 ($\mathsf{E\_Vote}(\mathsf{Auth}, v)$). An expert casts his vote by encrypting his choice and authenticating to the BB.

(1) The expert i computes $\boldsymbol{c} \leftarrow \mathsf{EC.Enc}_{\mathsf{pk}_\mathsf{T}}(v)$ and generates $\tau \leftarrow \mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Prove}(\boldsymbol{c})$, which is a proof of plaintext knowledge of $\boldsymbol{c}$.
(2) The expert i uses Auth to generate an authencation message e_i on $\boldsymbol{c}$.
(3) The expert i sends the ballot $\beta_\mathsf{E} = (\mathsf{E}_i, \boldsymbol{c}, \tau, e_i)$ to the BB.

Procedure 7 ($\mathsf{Validate}(\mathsf{BB}, \beta)$). The validation algorithm verifies that a ballot is valid with respect to the current state of the bulletin board.

(1) If B is a voter's ballot, parse β as $(\mathsf{pk}, \boldsymbol{u}, \pi, \sigma)$. Check (i) pk does not appear in other ballots; (ii) $\boldsymbol{u}$ does not appear in other ballots; (iii) $\mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Verify}(\pi, \boldsymbol{u}) \overset{?}{=} 1$; and (iv) $\mathsf{Sig.verify}_{\mathsf{pk}}(\sigma, \boldsymbol{u}) \overset{?}{=} 1$.
(2) If B is an expert's ballot, parse β as $(\mathsf{E}_i, \boldsymbol{c}, \tau, e_i)$. Check (i) E_i does not appear in other ballots; (ii) $\boldsymbol{c}$ does not appear in other ballots; (iii) $\mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Verify}(\tau, \boldsymbol{c}) \overset{?}{=} 1$; and (iv) e_i is E_i's valid authentication message on $\boldsymbol{c}$.
(3) Return $\top$ if and only if all the checks pass.

Tally Phase:

Procedure 8 (Tally). When the voting phase ends, the shuffler and trustees will perform the tally procedure.

(1) The shuffler takes all the registration items from the BB.
(2) For each registration item $\langle \tilde{K}, A, \mathsf{tx}, \delta, \sigma_{\mathsf{RA}}\rangle$ published by the RA, check (i) $\mathsf{NIZK}_{\mathsf{power}}.\mathsf{Verify}(A, \mathsf{tx}, \delta) \overset{?}{=} 1$; (ii) $\mathsf{Sig.Verify}_{\mathsf{pk}_{\mathsf{RA}}}(\sigma_{\mathsf{RA}}, \tilde{K}||A||\mathsf{tx}||\delta) \overset{?}{=} 1$. The shuffler removes the invalid ones and removes $\mathsf{tx}, \delta, \sigma_{\mathsf{RA}}$.
(3) For each fake registration item $\langle K', A_0, \rho\rangle$ published by a voter, check $\mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Verify}(K', \rho) \overset{?}{=} 1$. Remove the invalid ones and remove ρ.

(4) The shuffler takes all the ballots from the BB.
(5) For each voter's ballot $\beta_\mathsf{V} = (\mathsf{pk}, \boldsymbol{u}, \pi, \sigma)$, check (i) $\mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Verify}(\boldsymbol{u}, \pi) \overset{?}{=} 1$; (ii) $\mathsf{Sig.Verify}_{\mathsf{pk}}(\sigma, \boldsymbol{u}) \overset{?}{=} 1$. Remove the invalid ballots and remove π, σ. Now, a voter's ballot B is of the form $(\mathsf{pk}, \boldsymbol{u})$.
(6) For each expert's ballot $\beta_\mathsf{E} = (\mathsf{E}_i, \boldsymbol{c}, \tau, e_i)$, check (i) $\mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Verify}(\boldsymbol{c}, \tau) \overset{?}{=} 1$; (ii) e_i is a valid authentication message on $\boldsymbol{c}$. Remove the invalid ballots and remove τ, e_i. Now, an expert's ballot B_E is of the form $(\mathsf{E}_i, \boldsymbol{c})$.
(7) The shuffler puts all the valid registration items together. At this point, each registration item is of the form (K, A), where K is the encrypted public key, and A is the encrypted voting power.
(8) The shuffler verifiably shuffle re-encrypts the registration items.
(9) For each shuffled registration item (K, A), the trustees verifiably decrypt the public keys K. If the same public key appears multiple times, drop them.
(10) For each ballot B, if the public key does not match any public key in the registration items, drop it.
(11) Put the corresponding A together with $\boldsymbol{u}$, i.e., assume a ballot $B = (\mathsf{pk}, \boldsymbol{u})$ and a registration item $W = (K, A)$, if $\mathsf{EC.Dec}_{\mathsf{sk_T}}(K) = \mathsf{pk}$, we put A and $\boldsymbol{u}$ together to form a new item $I := (A, \boldsymbol{u})$.
(12) The shuffler verifiably shuffle re-encrypts the new items.
(13) For each candidate C_i, set initial score as $\boldsymbol{s}_i := \mathsf{LE.Enc}_{\mathsf{pk_T}}(0)$. For each expert E_j, set initial score as $\boldsymbol{s}_{\ell+j} := \mathsf{LE.Enc}_{\mathsf{pk_T}}(0)$.
(14) For each shuffled item $I := (A, \boldsymbol{u})$, the trustees verifiably decrypt $\boldsymbol{u}$ to v and update candidate (or expert) v's score $\boldsymbol{s}_v := \boldsymbol{s}_v \cdot A$.
(15) Form a list of each expert's encrypted score and encrypted candidate, i.e., expert E_i's entry is $(\boldsymbol{s}_{\ell+i}, \boldsymbol{c}_i)$.
(16) For each expert's ballot $(\boldsymbol{s}, \boldsymbol{c})$, the trustees verifiably decrypt $\boldsymbol{c}$ to v and update candidate v's score $\boldsymbol{s}_v := \boldsymbol{s}_v \cdot \boldsymbol{s}$.
(17) For each candidate C_i, the trustees verifiably decrypt the score $\boldsymbol{s}_i$ to s_i.
(18) Return $\{s_i\}_{i\in[\ell]}$ as the final result.

Procedure 9 (VerifyElection). Any party can verify that the election is performed correctly.

(1) Verify all experts' registration messages to check that the list of eligible experts is correctly formed.
(2) For each registration item published by the RA, parse it as $\langle \tilde{K}, A, \mathsf{tx}, \delta, \sigma_{\mathsf{RA}} \rangle$, and check (i) $\mathsf{NIZK}_{\mathsf{power}}.\mathsf{Verify}(A, \mathsf{tx}, \delta) \overset{?}{=} 1$; (ii) $\mathsf{Sig.Verify}_{\mathsf{pk_{RA}}}(\sigma_{\mathsf{RA}}, \tilde{K}||A||\mathsf{tx}||\delta) \overset{?}{=} 1$. Check that all the invalid ones are removed and all the valid ones are counted.
(3) For each fake registration item published by a voter, parse it as $\langle K', A_0, \rho \rangle$, check $\mathsf{NIZK}_{\mathsf{knowledge}}.\mathsf{Verify}(K', \rho) \overset{?}{=} 1$. Check that all the invalid ones are removed and all the valid ones are counted.
(4) Check all the shuffle NIZKs and decryption NIZKs during the tally phase.
(5) Return $\top$ if and only if all of the checks pass.

```
$\underline{\mathrm{Exp}^{BP}_{\mathcal{A},\mathcal{V}}[b](\lambda, C):}$
    $((\mathsf{pk_{RA}}, \mathsf{sk_{RA}}), (\mathsf{pk_T}, \mathsf{sk_T})) \leftarrow \mathsf{Setup}(1^\lambda, C)$;
    $b' \leftarrow \mathcal{A}^{O}(\mathsf{sk_{RA}}, \mathsf{pk_T}, C)$;
    output $b'$.
**Oracles:**
$O$voteLR$(\mathsf{pk}, \mathsf{sk}, v_0, v_1)$ :
    let $\beta_0 = \mathsf{V\_vote}(\mathsf{pk}, \mathsf{sk}, v_0)$ and $\beta_1 = \mathsf{V\_vote}(\mathsf{pk}, \mathsf{sk}, v_1)$;
    if $\mathsf{Validate}(\mathsf{BB}_b, \beta_b) = \perp$ then return $\perp$;
    else $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0||\beta_0$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1||\beta_1$.
$O$cast$(\beta)$ :
    if $\mathsf{Validate}(\mathsf{BB}_b, \beta) = \perp$ then return $\perp$;
    else $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0||\beta$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1||\beta$.
$O$board() :
    return $\mathsf{BB}_b$.
$O$tally() :
    $(r, \Pi_0) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathsf{sk_T})$;
    $\Pi_1 = \mathsf{SimTally}(\mathsf{BB}_1, r)$;
    return $(r, \Pi_b)$.
```

Figure 8: Ballot Privacy Experiment $\mathrm{Exp}^{BP}_{\mathcal{A},\mathcal{V}}[b](\lambda, C)$

5 SECURITY ANALYSIS

In this section, we analyze ballot privacy, verifiability, and coercion-resistance of our voting scheme. As mentioned above, experts do not have ballot privacy and coercion-resistance, so we can adopt the definitions of ballot privacy and coercion-resistance by Bernhard *et al.* [10, 11]. We adapt Cortier *et al.*'s verifiability definition [21] to delegative and weighted voting.

5.1 Ballot Privacy

Ballot privacy is based on the definition by Bernhard *et al.* [10]. The ballot privacy experiment tracks two bulletin boards: BB_0 for the "real" world and BB_1 for the "fake" world, in which only one bulletin board is available the the adversary $\mathcal{A}$. The outcome of the election is always computed on the real bulletin board BB_0. The adversary controls the registration authority RA and a subset of voters, and his goal is to distinguish whether the tally result is evaluated over the "real" or "fake" world.

Formally, the ballot privacy experiment is depicted in Figure 8. The adversary $\mathcal{A}$ takes as input RA's secret key $\mathsf{sk_{RA}}$ (since he controls the RA), the trustees' public key $\mathsf{pk_T}$, and the candidate list C, and he can query the following oracles:

- OvoteLR$(\mathsf{pk}, \mathsf{sk}, v_0, v_1)$, which allows the adversary $\mathcal{A}$ to let a voter with $(\mathsf{pk}, \mathsf{sk})$ cast a vote for v_0 on BB_0 and a vote for v_1 on BB_1.
- Ocast(β), which allows the adversary $\mathcal{A}$ to cast a ballot β (constructed by $\mathcal{A}$) on both BB_0 and BB_1.
- Oboard(), which allows the adversary $\mathcal{A}$ to see the content of a bulletin board depending on the bit b.
- Otally(), which allows the adversary $\mathcal{A}$ to see the tally result and proof of tally correctness. The result r is always evaluated by tallying BB_0, and the proof is simulated when $b = 1$.

Definition 1. *Ballot privacy. Let* $\mathcal{V}$ = {Setup, Reg, FakeReg, E_reg, V_vote, E_vote, Validate, Tally, VerifyElection} *be an election scheme for a candidate list* C *and security parameter* λ*. We say that* $\mathcal{V}$ *meets ballot privacy if there exists a PPT simulation algorithm* SimTally *such that for any PPT adversary* $\mathcal{A}$*:*

$$|\Pr[\mathrm{Exp}^{BP}_{\mathcal{A},\mathcal{V}}[0](\lambda, C) = 1] - \Pr[\mathrm{Exp}^{BP}_{\mathcal{A},\mathcal{V}}[1](\lambda, C) = 1]|$$

is a negligible function in the security parameter λ*.*

Theorem 1. *Assume that the ElGamal encryption scheme is IND-CPA secure, the underlying NIZKs* NIZK_i, $i \in$ {power, knowledge, DVP-reenc, sk, shuffle, Dec} *are complete, sound, and zero-knowledge,* $\mathsf{NIZK}_{\mathsf{knowledge}}$ *is simulation-sound extractable, and the signature scheme is EUF-CMA secure. The voting protocol described in secion 4 provides ballot privacy.*

The proof of this theorem is deferred to Appendix C. We also prove that our scheme provides strong consistency and strong correctness [10].

5.2 Verifiability

We adapt Cortier *et al.*'s verifiability definition [21] to our delegative and weighted voting. In a nutshell, a voting scheme is verifiable if the election result reflects the votes of 1) all honest voters/experts who checked their votes, which appear on the bulletin board; 2) at most n_c voters/experts who are controlled by the adversary; and 3) a subset of the votes by honest voters/experts that did not check if their ballots were cast correctly. In our scheme, the registration authority is not trusted for verifiability, which is known as strong verifiability [21].

The verifiability experiment is formally depicted in Figure 9. We use $\mathbb{U}$ to denote the set of public-private key pairs and Corrupted to denote the set of corrupted voters/experts. Let $H = \{(\mathsf{pk}_i^h, v_i^h, \alpha_i^h, *)\}_{i=1}^{n_h}$ correspond to voters/experts that have checked that their ballots will be counted and $H' = \{(\mathsf{pk}_i^{h'}, v_i^{h'}, \alpha_i^{h'}, *)\}_{i=1}^{n_{h'}}$ correspond to voters/experts that have not checked their ballots (for experts, $\alpha_i = \emptyset$). We adapt the result function ρ to the delegative and weighted voting setting, i.e., $\rho : (\mathbb{V} \vee \mathbb{E})^{\tau} \longrightarrow \boldsymbol{R}$ takes as inputs voters' and experts' choices, and outputs the election result. The adversary $\mathcal{A}$ takes as input RA's secret key $\mathsf{sk}_{\mathsf{RA}}$, the trustees' secret key sk_{T}, and the candidate list C, and he can query the following oracles:

- $\mathcal{O}$corrupt(id), which allows the adversary $\mathcal{A}$ to corrupt a voter/expert id.
- $\mathcal{O}$vote(id, v), which allows the adversary $\mathcal{A}$ to let an honest voter/expert id cast a vote for v.

Note that $\mathcal{A}$ does not need the $\mathcal{O}$register oracle since he controls the registration authority and thus can register arbitrarily. The adversary wins if the result r verifies but violates any of the following conditions: 1) for each voter/expert that has checked his ballot, the ballot is counted; 2) at most n_c corrupted votes are counted; 3) a subset of votes by honest voters/experts that did not check their ballots are counted.

Definition 2. *Verifiability. Let* $\mathcal{V}$ = {Setup, Reg, FakeReg, E_reg, V_vote, E_vote, Validate, Tally, VerifyElection} *be an election scheme for a candidate list* C *and security parameter* λ*. We say that* $\mathcal{V}$ *meets verifiability if for any PPT adversary* $\mathcal{A}$*:*

$$\Pr[\mathrm{Exp}^{ver}_{\mathcal{A},\mathcal{V}}(\lambda, C) = \top]$$

```
$\underline{\mathrm{Exp}^{ver}_{\mathcal{A},\mathcal{V}}(\lambda, C)}$ :
    $((\mathsf{pk_{RA}}, \mathsf{sk_{RA}}), (\mathsf{pk_T}, \mathsf{sk_T})) \leftarrow \mathsf{Setup}(1^\lambda, C)$;
    $(\mathsf{BB}, r, \Pi) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{sk_{RA}}, \mathsf{sk_T})$;
    if $\mathsf{VerifyElection}(\mathsf{BB}, (r, \Pi)) = \perp$ then return $\perp$;
    if $r = \perp$ then return $\perp$;
    $H = \{(\mathsf{pk}_i^h, v_i^h, \alpha_i^h, *)\}_{i=1}^{n_h}$ and $H' = \{(\mathsf{pk}_i^{h'}, v_i^{h'}, \alpha_i^{h'}, *)\}_{i=1}^{n_{h'}}$;
    if $\exists\{\mathsf{pk}_i^c, v_i^c, \alpha_i^c\}_{i=1}^{n_c} \subseteq$ Corrupted, $\exists\{(\mathsf{pk}_i^{h'}, v_i^{h'}, \alpha_i^{h'}, *)\}_{i=1}^{t} \subseteq H'$ s.t.
        $r = \rho(\{\mathsf{pk}_i^h, v_i^h, \alpha_i^h\}_{i=1}^{n_h} \cup \{\mathsf{pk}_i^c, v_i^c, \alpha_i^c\}_{i=1}^{n_c} \cup \{\mathsf{pk}_i^{h'}, v_i^{h'}, \alpha_i^{h'}\}_{i=1}^{t})$
    then return $\perp$ else return $\top$.
**Oracles:**
$\mathcal{O}$corrupt($id$) :
    if $(id, *, *) \notin \mathbb{U}$ then return $\perp$;
    Corrupted $\leftarrow$ Corrupted $\cup \{(\mathsf{pk}_{id}, *, \alpha_{id})\}$;
    return $(\mathsf{pk}_{id}, \mathsf{sk}_{id})$;
$\mathcal{O}$vote($id, v$) :
    if $(id, *, *) \notin \mathbb{U}$ or $(id, *) \in$ Corrupted then return $\perp$;
    if $id$ is voter then
        $\beta \leftarrow \mathsf{V\_vote}(\mathsf{pk}_{id}, \mathsf{sk}_{id}, v)$;
    else
        $\beta \leftarrow \mathsf{E\_vote}(\mathsf{Auth}_{id}, v)$;
    end if
    $H \cup H' \leftarrow \mathsf{Update}(H \cup H', (id, v, \alpha_{id}, \beta))$;
    return $\beta$.
```

Figure 9: Verifiability Experiment $\mathrm{Exp}^{ver}_{\mathcal{A},\mathcal{V}}(\lambda, C)$

is a negligible function in the security parameter λ*.*

Theorem 2. *Assume that the underlying NIZKs* NIZK_i, $i \in$ {power, knowledge, DVP-reenc, sk, shuffle, Dec} *are complete, sound, and zero-knowledge, and the signature scheme is EUF-CMA secure. The voting protocol described in section 4 provides verifiability.*

The proof of this theorem is deferred to Appendix D.

5.3 Coercion-resistance

The definition of coercion-resistance is inspired by Bernhard *et al.* [11]. Similar to the ballot privacy definition, the coercion-resistance experiment tracks two bulletin boards for the "real" world and the "fake" world, respectively. The adversary's goal is to distinguish whether he is in the "real" or "fake" world.

The coercion-resistance experiment is depicted in Figure 10. The adversary $\mathcal{A}$ takes as input RA's public key $\mathsf{pk}_{\mathsf{RA}}$, the trustees' public key sk_{T}, and the candidate list C (since RA and trustees are trusted for coercion-resistance). The adversary has access to $\mathcal{O}$voteLR, $\mathcal{O}$cast, $\mathcal{O}$board, and $\mathcal{O}$tally as in the ballot privacy experiment, along with an additional oracle $\mathcal{O}$receipt(id, v_0, v_1). $\mathcal{O}$receipt (id, v_0, v_1) allows the adversary $\mathcal{A}$ to let a voter id submit to coercion by voting for v_0 on BB_0, and resist coercion by voting for v_1 on BB_1. It will return the ballot cast under coercion β_b and a receipt. If the voter submits to coercion, the receipt is the real view during the registration and voting phase; if the voter resists coercion, the receipt is simulated by the algorithm SimView.

Definition 3. *Coercion-resistance. Let* $\mathcal{V}$ = {Setup, Reg, FakeReg, E_reg, V_vote, E_vote, Validate, Tally, VerifyElection} *be an election*

```
$\underline{\mathrm{Exp}^{CR}_{\mathcal{A},\mathcal{V}}[b](\lambda, C):}$
  $((\mathsf{pk_{RA}}, \mathsf{sk_{RA}}), (\mathsf{pk_T}, \mathsf{sk_T})) \leftarrow \mathsf{Setup}(1^\lambda, C)$;
  $b' \leftarrow \mathcal{A}^{O}(\mathsf{pk_{RA}}, \mathsf{pk_T}, C)$;
  Output $b'$.
**Oracles:**
$O$voteLR$(id, v_0, v_1)$ :
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Reg}(id)$;
  let $\beta_0 = \mathsf{V\_vote}(\mathsf{pk}, \mathsf{sk}, v_0)$ and $\beta_1 = \mathsf{V\_vote}(\mathsf{pk}, \mathsf{sk}, v_1)$;
  if $\mathsf{Validate}(\mathsf{BB}_b, \beta_b) = \perp$ then return $\perp$;
  else $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 || \beta_0$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 || \beta_1$.
$O$receipt$(id, v_0, v_1)$ :                    % submit versus resist
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Reg}(id)$;
  $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{FakeReg}(id)$;
  let $\beta_0 = \mathsf{V\_vote}(\mathsf{pk}, \mathsf{sk}, v_0)$ and $\beta_1 = \mathsf{V\_vote}(\mathsf{pk}', \mathsf{sk}', v_0)$;
  if $\mathsf{Validate}(\mathsf{BB}_b, \beta_b) = \perp$ then return $\perp$;
  else $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 || \beta_0$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 || \beta_1$.
  if $b = 0$ then
    receipt = $\mathsf{View}\{\mathsf{Reg}(id), \mathsf{V\_vote}(\mathsf{pk}, \mathsf{sk}, v_0)\}$;
  else
    receipt = $\mathsf{SimView}(\mathsf{sk}_{id})$;
  end if
  let $\beta'_0 = \mathsf{V\_vote}(\mathsf{pk}', \mathsf{sk}', v_1)$ and $\beta'_1 = \mathsf{V\_vote}(\mathsf{pk}, \mathsf{sk}, v_1)$;
  if $\mathsf{Validate}(\mathsf{BB}_b, \beta'_b) = \perp$ then return $\perp$;
  else $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 || \beta'_0$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 || \beta'_1$.
  return $(\beta_b, \text{receipt})$.
$O$cast$(\beta)$ :
  if $\mathsf{Validate}(BB_b, \beta) = \perp$ then return $\perp$;
  else $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 || \beta$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 || \beta$.
$O$board() :
  return $\mathsf{BB}_b$.
$O$tally() :
  $(r, \Pi_0) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathsf{sk_T})$;
  $\Pi_1 = \mathsf{SimTally}(\mathsf{BB}_1, r)$;
  return $(r, \Pi_b)$.
```

Figure 10: Coercion-resistance Experiment $\mathrm{Exp}^{CR}_{\mathcal{A},\mathcal{V}}[b](\lambda, C)$

scheme for a candidate list C and security parameter λ. We say that $\mathcal{V}$ meets coercion-resistance if there exist PPT simulation algorithms SimView *and* SimTally *such that for any PPT adversary $\mathcal{A}$:*

$$|\Pr[\mathrm{Exp}^{CR}_{\mathcal{A},\mathcal{V}}[0](\lambda, C) = 1] - \Pr[\mathrm{Exp}^{CR}_{\mathcal{A},\mathcal{V}}[1](\lambda, C) = 1]|$$

is a negligible function in the security parameter λ.

Theorem 3. *Assume that the ElGamal encryption scheme is IND-CPA secure, the underlying NIZKs NIZK_i, $i \in$ {power, knowledge, DVP-reenc, sk, shuffle, Dec} are complete, sound, and zero-knowledge, $\mathsf{NIZK_{knowledge}}$, $\mathsf{NIZK_{sk}}$ are simulation-sound extractable, and the signature scheme is EUF-CMA secure. The voting protocol described in section 4 provides coercion-resistance.*

The proof of this theorem is deferred to Appendix E.

6 DISCUSSION

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 effectively prevents vote-buying on the blockchain.

Stake renting/smart contract vote-buying. Another attack on blockchain voting is to use a smart contract to "rent" stakes. The smart contract collects stakes, uses the stakes for voting, and returns them back with an extra payment after the election. We defend against this attack by prohibiting contract accounts from participating in the voting.

Inalienable authentication. Coercion-resistant voting requires inalienable authentication [1], i.e., the coercer can neither impersonate the voter nor prevent the voter from authenticating. In the blockchain context, authentication is realized by proving knowledge of his blockchain secret key, and it is assumed that the voter will not give his secret key to the coercer. However, the "Dark DAO" attack [5, 24] is still possible if the coercer can use TEEs. Specifically, the coercer can set up a TEE running a "cryptocurrency wallet" and create an encumbered key. The remote attestation of the TEE can prove that the wallet will not steal the bribee's stake. Then, a voter can transfer his stake to the account and use his own secret key to manage it. At the end of the election, the TEE ensures a fair exchange for voting-buying. As pointed out in [24], this problem is inherent in any remote voting scheme where the secret key is generated by the voter. Kelkar *et al.* [35] proposed two schemes to defend against such kind of attacks using TEEs and ASICs (Application-Specific Integrated Circuit), respectively. But neither of them is suitable in the blockchain context. In this work, we assume that authentication is inalienable. How to defend against this type of attack is out of the scope of this paper. We leave it as an interesting open question.

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*, the ballot generation has $O(1)$ complexity. In the *tally phase*, the shuffler shuffles the ballots and the registration 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 the voting power and candidates, and decrypt the candidates. Thus, the time complexity of a trustee is also $O(n)$. Adding them together, our scheme has $O(n)$ time complexity.

7 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

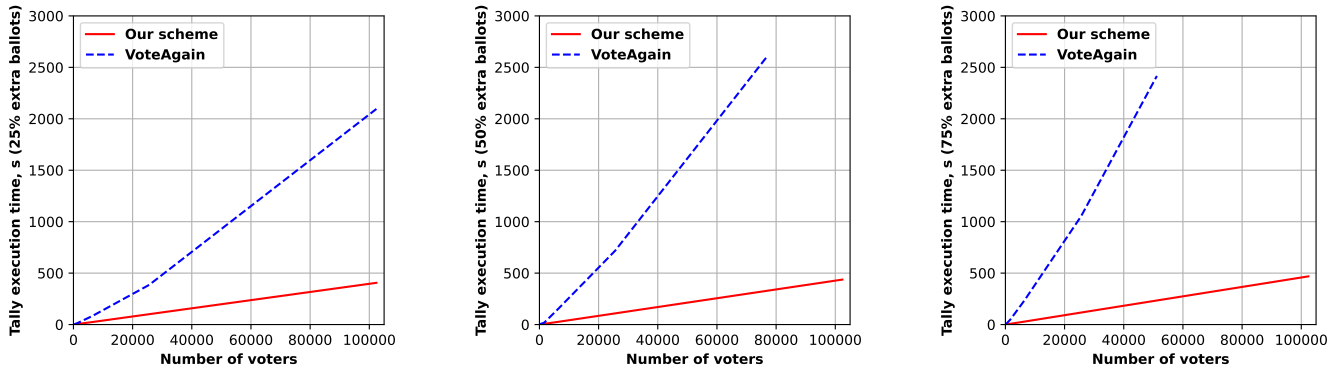


Figure 11: Comparison of tally execution time between our scheme and VoteAgain [40] (with extra ballot rate as 25%, 50%, 75% from left to right). In VoteAgain, x% extra ballot rate means that x% voters re-vote once; in our scheme, x% extra ballot rate means that x% voters cast one fake ballot.

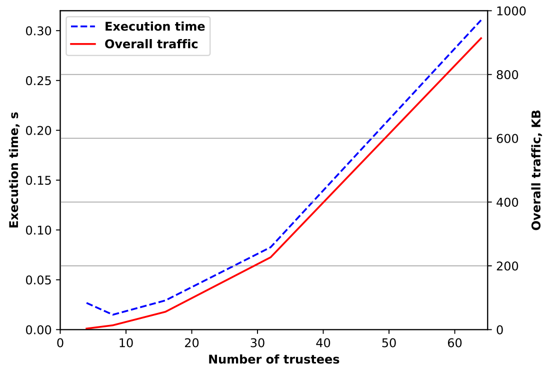


Figure 12: DKG execution time and overall traffic with respect to different numbers of trustees

curve math, and it uses Schnorr signature as the signature scheme. We evaluate 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 Figure 12.

Registration phase. In the registration phase, the voter uses $\mathsf{NIZK}_{\mathsf{power}}$ in the registration message to prove that the frozen stake is equal to the encrypted voting power, and the RA proves re-encryption correctness by a designated verifier proof $\mathsf{NIZK}_{\mathsf{DVP\text{-}reenc}}$. Generating a registration message 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 registration 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, sign it, and use $\mathsf{NIZK}_{\mathsf{knowledge}}$ 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 [40]. 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.

Figure 11 shows the tally execution time compared with VoteAgain [40] 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 rates (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.

8 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 that the proposed scheme has ballot privacy, verifiability, and coercion-resistance without any extra strong assumptions. Our scheme has an advantage over all the existing coercion-resistant voting schemes, so it is suitable for large-scale coercion-resistant blockchain voting programs.

REFERENCES

[1] Dirk Achenbach, Carmen Kempka, Bernhard Löwe, and Jörn Müller-Quade. 2015. Improved Coercion-Resistant Electronic Elections through Deniable Re-Voting. *USENIX Journal of Election Technology and Systems (JETS)* (Aug. 2015). https://www.usenix.org/conference/jets15/workshop-program/presentation/achenbach

[2] Ben Adida. 2008. Helios: Web-based Open-Audit Voting. In *USENIX Security 2008: 17th USENIX Security Symposium*, Paul C. van Oorschot (Ed.). USENIX Association, San Jose, CA, USA, 335–348.

[3] Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. 2015. Incoercible Multi-party Computation and Universally Composable Receipt-Free Voting. In *Advances in Cryptology – CRYPTO 2015, Part II (Lecture Notes in Computer Science, Vol. 9216)*, Rosario Gennaro and Matthew J. B. Robshaw (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 763–780. https://doi.org/10.1007/978-3-662-48000-7_37

[4] Roberto Araújo, Amira Barki, Solenn Brunet, and Jacques Traoré. 2016. Remote Electronic Voting Can Be Efficient, Verifiable and Coercion-Resistant. In *FC 2016 Workshops (Lecture Notes in Computer Science, Vol. 9604)*, Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff (Eds.). Springer, Heidelberg, Germany, Christ Church, Barbados, 224–232. https://doi.org/10.1007/978-3-662-53357-4_15

[5] James Austgen, Andrés Fábrega, Sarah Allen, Kushal Babel, Mahimna Kelkar, and Ari Juels. 2023. DAO Decentralization: Voting-Bloc Entropy, Bribery, and Dark DAOs. *CoRR* abs/2311.03530 (2023). https://doi.org/10.48550/ARXIV.2311.03530 arXiv:2311.03530

[6] Stephanie Bayer and Jens Groth. 2012. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In *Advances in Cryptology – EUROCRYPT 2012 (Lecture Notes in Computer Science, Vol. 7237)*, David Pointcheval and Thomas Johansson (Eds.). Springer, Heidelberg, Germany, Cambridge, UK, 263–280. https://doi.org/10.1007/978-3-642-29011-4_17

[7] Mihir Bellare and Phillip Rogaway. 1994. Entity Authentication and Key Distribution. In *Advances in Cryptology – CRYPTO'93 (Lecture Notes in Computer Science, Vol. 773)*, Douglas R. Stinson (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 232–249. https://doi.org/10.1007/3-540-48329-2_21

[8] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, Berkeley, CA, USA, 459–474. https://doi.org/10.1109/SP.2014.36

[9] Juan Benet. 2014. IPFS - Content Addressed, Versioned, P2P File System. *CoRR* abs/1407.3561 (2014). arXiv:1407.3561 http://arxiv.org/abs/1407.3561

[10] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. 2015. SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 499–516. https://doi.org/10.1109/SP.2015.37

[11] David Bernhard, Oksana Kulyk, and Melanie Volkamer. 2017. Security Proofs for Participation Privacy, Receipt-Freeness and Ballot Privacy for the Helios Voting Scheme. In *Proceedings of the 12th International Conference on Availability, Reliability and Security, Reggio Calabria, Italy, August 29 - September 01, 2017*. ACM, 1:1–1:10. https://doi.org/10.1145/3098954.3098990

[12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. 2012. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In *Advances in Cryptology – ASIACRYPT 2012 (Lecture Notes in Computer Science, Vol. 7658)*, Xiaoyun Wang and Kazue Sako (Eds.). Springer, Heidelberg, Germany, Beijing, China, 626–643. https://doi.org/10.1007/978-3-642-34961-4_38

[13] Alex Biryukov and Ivan Pustogarov. 2015. Bitcoin over Tor isn't a Good Idea. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 122–134. https://doi.org/10.1109/SP.2015.15

[14] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 315–334. https://doi.org/10.1109/SP.2018.00020

[15] Sergiu Bursuc, Gurchetan S. Grewal, and Mark Dermot Ryan. 2011. Trivitas: Voters Directly Verifying Votes. In *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 7187)*, Aggelos Kiayias and Helger Lipmaa (Eds.). Springer, 190–207. https://doi.org/10.1007/978-3-642-32747-6_12

[16] David Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* 24, 2 (1981), 84–88. https://doi.org/10.1145/358549.358563

[17] David Chaum. 2016. Random-sample voting. *White Paper* (2016).

[18] Jing Chen and Silvio Micali. 2019. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.* 777 (2019), 155–183. https://doi.org/10.1016/J.TCS.2019.02.001

[19] Jeremy Clark and Urs Hengartner. 2012. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. In *FC 2011: 15th International Conference on Financial Cryptography and Data Security (Lecture Notes in Computer Science, Vol. 7035)*, George Danezis (Ed.). Springer, Heidelberg, Germany, Gros Islet, St. Lucia, 47–61.

[20] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. 2008. Civitas: Toward a Secure Voting System. In *2008 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, Oakland, CA, USA, 354–368. https://doi.org/10.1109/SP.2008.32

[21] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. 2014. Election Verifiability for Helios under Weaker Trust Assumptions. In *ESORICS 2014: 19th European Symposium on Research in Computer Security, Part II (Lecture Notes in Computer Science, Vol. 8713)*, Miroslaw Kutylowski and Jaideep Vaidya (Eds.). Springer, Heidelberg, Germany, Wroclaw, Poland, 327–344. https://doi.org/10.1007/978-3-319-11212-1_19

[22] Véronique Cortier and Joseph Lallemand. 2018. Voting: You Can't Have Privacy without Individual Verifiability. In *ACM CCS 2018: 25th Conference on Computer and Communications Security*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, Toronto, ON, Canada, 53–66. https://doi.org/10.1145/3243734.3243762

[23] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. 1994. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Advances in Cryptology – CRYPTO'94 (Lecture Notes in Computer Science, Vol. 839)*, Yvo Desmedt (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 174–187. https://doi.org/10.1007/3-540-48658-5_19

[24] Philip Daian, Tyler Kell, Ian Miers, and Ari Juels. 2018. *On-Chain Vote Buying and the Rise of Dark DAOs.* https://hackingdistributed.com/2018/07/02/on-chain-vote-buying (Last accessed: 2024-04-18).

[25] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology – CRYPTO'86 (Lecture Notes in Computer Science, Vol. 263)*, Andrew M. Odlyzko (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 186–194. https://doi.org/10.1007/3-540-47721-7_12

[26] Bryan Alexander Ford. 2002. Delegative democracy. (2002).

[27] Chaya Ganesh, Claudio Orlandi, and Daniel Tschudi. 2019. Proof-of-Stake Protocols for Privacy-Aware Blockchains. In *Advances in Cryptology – EUROCRYPT 2019, Part I (Lecture Notes in Computer Science, Vol. 11476)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, Heidelberg, Germany, Darmstadt, Germany, 690–719. https://doi.org/10.1007/978-3-030-17653-2_23

[28] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *Advances in Cryptology – EUROCRYPT'99 (Lecture Notes in Computer Science, Vol. 1592)*, Jacques Stern (Ed.). Springer, Heidelberg, Germany, Prague, Czech Republic, 295–310. https://doi.org/10.1007/3-540-48910-X_21

[29] Rosario Giustolisi, Maryam Sheikhi Garjan, and Carsten Schuermann. 2023. Thwarting Last-Minute Voter Coercion. Cryptology ePrint Archive, Paper 2023/1876. https://eprint.iacr.org/2023/1876 https://eprint.iacr.org/2023/1876.

[30] Kristian Gjøsteen. 2011. The Norwegian Internet Voting Protocol. In *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 7187)*, Aggelos Kiayias and Helger Lipmaa (Eds.). Springer, 1–18. https://doi.org/10.1007/978-3-642-32747-6_1

[31] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology – EUROCRYPT 2016, Part II (Lecture Notes in Computer Science, Vol. 9666)*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer, Heidelberg, Germany, Vienna, Austria, 305–326. https://doi.org/10.1007/978-3-662-49896-5_11

[32] Lucca Hirschi, Lara Schmid, and David A. Basin. 2021. Fixing the Achilles Heel of E-Voting: The Bulletin Board. In *CSF 2021: IEEE 34th Computer Security Foundations Symposium*, Ralf Küsters and Dave Naumann (Eds.). IEEE Computer Society Press, Virtual Conference, 1–17. https://doi.org/10.1109/CSF51468.2021.00016

[33] Martin Hirt and Kazue Sako. 2000. Efficient Receipt-Free Voting Based on Homomorphic Encryption. In *Advances in Cryptology – EUROCRYPT 2000 (Lecture Notes in Computer Science, Vol. 1807)*, Bart Preneel (Ed.). Springer, Heidelberg, Germany, Bruges, Belgium, 539–556. https://doi.org/10.1007/3-540-45539-6_38

[34] Ari Juels, Dario Catalano, and Markus Jakobsson. 2005. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*, Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine (Eds.). ACM, 61–70. https://doi.org/10.1145/1102199.1102213

[35] Mahimna Kelkar, Kushal Babel, Philip Daian, James Austgen, Vitalik Buterin, and Ari Juels. 2023. Complete Knowledge: Preventing Encumbrance of Cryptographic Secrets. Cryptology ePrint Archive, Report 2023/044. https://eprint.iacr.org/2023/044.

[36] Thomas Kerber, Aggelos Kiayias, Markulf Kohlweiss, and Vassilis Zikas. 2019. Ouroboros Crypsinous: Privacy-Preserving Proof-of-Stake. In *2019 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 157–174. https://doi.org/10.1109/SP.2019.00063

[37] Markulf Kohlweiss, Varun Madathil, Kartik Nayak, and Alessandra Scafuro. 2021. On the Anonymity Guarantees of Anonymous Proof-of-Stake Protocols. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 1818–1833. https://doi.org/10.1109/SP40001.2021.00107

[38] Yannan Li, Willy Susilo, Guomin Yang, Yong Yu, Dongxi Liu, Xiaojiang Du, and Mohsen Guizani. 2022. A Blockchain-Based Self-Tallying Voting Protocol in Decentralized IoT. *IEEE Trans. Dependable Secur. Comput.* 19, 1 (2022), 119–130. https://doi.org/10.1109/TDSC.2020.2979856

[39] Philipp Locher, Rolf Haenni, and Reto E. Koenig. 2016. Coercion-Resistant Internet Voting with Everlasting Privacy. In *FC 2016 Workshops (Lecture Notes in Computer Science, Vol. 9604)*, Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff (Eds.). Springer, Heidelberg, Germany, Christ Church, Barbados, 161–175. https://doi.org/10.1007/978-3-662-53357-4_11

[40] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. 2020. VoteAgain: A scalable coercion-resistant voting system. In *USENIX Security 2020: 29th USENIX Security Symposium*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 1553–1570.

[41] Emmanouil Magkos, Mike Burmester, and Vassilios Chrissikopoulos. 2001. Receipt-Freeness in Large-Scale Elections without Untappable Channels. In *Towards The E-Society: E-Commerce, E-Business, and E-Government, The First IFIP Conference on E-Commerce, E-Business, E-Government (I3E 2001), October 3-5, Zürich, Switzerland (IFIP Conference Proceedings, Vol. 202)*, Beat F. Schmid, Katarina Stanoevska-Slabeva, and Volker Tschammer (Eds.). Kluwer, 683–693. https://doi.org/10.1007/0-306-47009-8_50

[42] Peter Y. A. Ryan, David Bismark, James Heather, Steve A. Schneider, and Zhe Xia. 2009. Prêt à voter: a voter-verifiable voting system. *IEEE Trans. Inf. Forensics Secur.* 4, 4 (2009), 662–673. https://doi.org/10.1109/TIFS.2009.2033233

[43] Snapshot. 2024. *Snapshot.* https://snapshot.org

[44] Votem. 2024. *Votem.* https://votem.com

[45] Bingsheng Zhang, Roman Oliynykov, and Hamed Balogun. 2019. A Treasury System for Cryptocurrencies: Enabling Better Collaborative Intelligence. In *ISOC Network and Distributed System Security Symposium – NDSS 2019.* The Internet Society, San Diego, CA, USA.

A DEFINITIONS

Here, we give formal definitions of Sigma protocols, Fiat-Shamir heuristic, public-key encryption, and signatures.

Sigma protocols. A Sigma protocol is a special type of 3-move public-coin proof system. A Sigma protocol for relation R consists of three algorithms $(\mathcal{C}, \mathcal{Z}, \mathcal{V})$:

- $a \leftarrow \mathcal{C}(x, w; r)$ takes as input the statement x, witness w and random coin r. It outputs the initial message a.
- $z \leftarrow \mathcal{Z}(x, w, r, e)$ takes as input the statement x, witness w, random coin r and a given challenge e. It outputs the answer z.
- $0/1 \leftarrow \mathcal{V}(x, a, e, z)$ takes as input the statement x, initial message a, challenge e and answer z. It outputs 0 or 1 for rejection or acceptance, respectively.

The completeness, k-special soundness, and special honest verifier zero-knowledge of Sigma protocols are defined as follows:

- Completeness: for all $(x, w) \in R, e \in \{0,1\}^\lambda$, $\Pr[a \leftarrow \mathcal{C}(x, w; r), z \leftarrow \mathcal{Z}(x, w, r, e) : \mathcal{V}(x, a, e, z) = 1] = 1$
- k-special soundness: there exists a deterministic polynomial-time extractor $\mathcal{X}$ such that for any PPT adversary $\mathcal{A}$: $\Pr[(x, a) \leftarrow \mathcal{A}(1^\lambda), e_1, \ldots, e_k \leftarrow \{0,1\}^\lambda, z_1, \ldots, z_k \leftarrow \mathcal{A}(e_1, \ldots, e_k), w \leftarrow \mathcal{X}(x, a, \{e_1, z_1\}, \ldots, \{e_k, z_k\}) : \mathcal{V}(x, a, e_i, z_i) = 1$ for $i \in [k] \wedge R(x, w) = 0] \approx 0$.
- Special honest verifier zero-knowledge: there exists a PPT algorithm $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$: $\Pr[(x, e) \leftarrow \mathcal{A}(1^\lambda); a \leftarrow \mathcal{C}(x, w; r); z \leftarrow \mathcal{Z}(x, w, r, e) : \mathcal{A}(a, z) = 1] \approx \Pr[(x, e) \leftarrow \mathcal{A}(1^\lambda); (a, z) \leftarrow \mathcal{S}(x, e) : \mathcal{A}(a, z) = 1]$

Fiat-Shamir Transformation [25]. Let $\Sigma = (\mathcal{C}, \mathcal{Z}, \mathcal{V})$ be a Sigma protocol and $\mathcal{H}$ a hash function. The Fiat-Sharmir transformation of Σ is the proof system $\mathsf{FS}_{\mathcal{H}}(\Sigma) = (\mathsf{Prove}, \mathsf{Verify})$ defined as follows:

- $(a, z) \leftarrow \mathsf{Prove}(x, w; r)$: Run $\mathcal{C}(x, w; r)$ to obtain initial message a. Compute $e \leftarrow \mathcal{H}(x, a)$. Run $\mathcal{Z}(x, w, r, e)$ to get the answer z. Output (a, z).
- $0/1 \leftarrow \mathsf{Verify}(x, a, z)$: Compute $e \leftarrow \mathcal{H}(x, a)$, then run $\mathcal{V}(x, a, e, z)$.

In the security proof, the hash function $\mathcal{H}$ is typically modeled as a random oracle. In the following, we give the definition of zero-knowledge [7] and simulation-sound extractability [12] (in the random oracle model).

Zero-knowledge [7]. A proof system $(\mathsf{Prove}, \mathsf{Verify})$ for relation $\mathcal{R}$ is zero-knowledge if there is a simulator $\mathcal{S}$ such that no adversary who can make queries to the random oracle and queries of the form $\mathsf{create\text{-}proof}(x, w)$ can distinguish the following two settings with non-negligibly better than 1/2 probability:

(1) Random oracle queries are answered by a random oracle. In response to $\mathsf{create\text{-}proof}(x, w)$, the challenger checks that $\mathcal{R}(x, w)$. If not, he returns $\perp$. Otherwise, he returns $\mathsf{Prove}(x, w)$.
(2) The challenger runs a copy of the simulator $\mathcal{S}$. It forwards random oracle queries to $\mathcal{S}$ directly. For $\mathsf{create\text{-}proof}(x, w)$, the challenger checks if $\mathcal{R}(x, w)$ holds: if not, the challenger returns $\perp$; if it holds, the challenger send $\mathsf{Simulate(x)}$ to $\mathcal{S}$ and returns the result to the adversary.

Simulation-sound extractability [12]. Let $\mathcal{P}$ be a zero-knowledge proof system with simulator $\mathcal{S}$. We say that $\mathcal{P}$ is simulation-sound extractable if there exists an extractor $\mathcal{K}$ such that for every prover $\mathcal{A}$, $\mathcal{K}$ wins the following game with overwhelming probability.

(1) (Initial run.) The game selects a random string ω for $\mathcal{A}$. It runs an instance of $\mathcal{A}$ with the simulator $\mathcal{S}$ until $\mathcal{A}$ makes his output and halts. If $\mathcal{A}$ does not output any proofs, any of the proofs do not verify (w.r.t. the instance of $\mathcal{S}$ used as the random oracle) or any of $\mathcal{A}$'s statement/proof pairs (x, π) is such that π was the result of a $\mathsf{Simulate}(x)$ query, then $\mathcal{K}$ wins the game directly.
(2) (Extraction.) The game runs an instance of $\mathcal{K}$, giving it the transcript of all queries in the initial run and the produced (x, π) as input. $\mathcal{K}$ may repeatedly make one type of query invoke in response to which the game runs a new invocation of $\mathcal{A}$ on the same randomness ω that it chose for the initial run. All queries made by these instances are forwarded to $\mathcal{K}$ who can reply to them.
(3) $\mathcal{K}$ wins the game if it can output a vector of witnesses w that match the statements x of the initial run, i.e. for all i we have $\mathcal{R}(x_i, w_i)$.

Public-key encryption. A public-key encryption scheme consists of three PPT algorithms $(\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$. The ElGamal encryption scheme we use is IND-CPA secure under the DDH assumption. Formally, consider the following IND-CPA experiment:

$\underline{\mathrm{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{Enc}}(\lambda):}$

(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.

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

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

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

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 $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig}}(\mathcal{A}, \lambda) := \Pr[\mathrm{Exp}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sig}}(\lambda) = 1]$ is negligible in λ.

B 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 knowledge ($\mathsf{NIZK}_{\mathsf{knowledge}}$); (iii) re-encryption correctness ($\mathsf{NIZK}_{\mathsf{DVP\text{-}reenc}}$); (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 [6] for shuffle correctness and Gennaro *et al.*'s scheme [28] for decryption correctness. Here, we demonstrate the corresponding Sigma protocols. In our scheme, they are transformed into NIZKs by Fiat-Shamir heuristic [25]. By the result of Bernhard *et al.* [12], Fiat-Shamir transformation on Sigma protocols yields simulation-sound extractability in the random oracle model.

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 encryption scheme, then Figure 13 shows the Sigma protocol for voting power correctness. If it is encrypted by a hybrid encryption scheme (e.g., ZCash [8]), then we can utilize other zero-knowledge protocols for general circuits (e.g. Groth16 [31], Bulletproofs [14]).

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 be proven with the Sigma protocol depicted in Figure 14.

Proof of re-encryption correctness. In the registration phase, the RA needs to generate a designated verifier proof 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 blockchain secret key" so that the verifier can simulate the proof. The Sigma protocol for re-encryption correctness is depicted in Figure 15. By the CDS composition [23], 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.

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 $\alpha', r_1', r_2' \leftarrow_\$ \mathbb{Z}_p$;
- 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 $e \leftarrow_\$ \mathbb{Z}_p$.

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 13: Sigma protocol for voting power correctness

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

Verifier:
- $V \rightarrow P$: random $e \leftarrow_\$ \mathbb{Z}_p$.

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 14: 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 $r' \leftarrow_\$ \mathbb{Z}_p$;
- Compute $a_1 = g^{r'}, a_2 := h^{r'}$;
- $P \longrightarrow V$: a_1, a_2.

Verifier:

- $V \longrightarrow P$: random $e \leftarrow_\$ \mathbb{Z}_p$.

Prover:

- Compute $z := r' + e \cdot r$;
- $P \longrightarrow V$: 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 15: 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 $r' \leftarrow_\$ \mathbb{Z}_p, x' \leftarrow_\$ \mathbb{G}$;
- Compute $a_1 := g^{r'}, a_2 := g^{x'} \cdot h^{r'}$;
- $P \longrightarrow V$: a_1, a_2.

Verifier:

- $V \longrightarrow P$: random $e \leftarrow_\$ \mathbb{Z}_p$.

Prover:

- Compute $z_1 := r' + e \cdot r, z_2 := x' + e \cdot x$;
- $P \longrightarrow V$: 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 16: Sigma protocol for knowledge of secret key

Proof of knowledge of secret key. To publish a fake registration item on the BB, the voter needs to prove knowledge of the corresponding secret key. This is a variant of the Schnorr protocol, depicted in Figure 16.

C PROOF OF BALLOT PRIVACY, STRONG CONSISTENCY, AND STRONG CORRECTNESS

In this section, we prove that our scheme provides ballot privacy. We also show that our scheme has strong consistency and strong correctness by Bernhard *et al.* [10], which are necessary for a voting scheme to guarantee ballot privacy.

C.1 Ballot privacy

Proof of theorem 1. To prove this theorem, we construct the SimTally algorithm and prove indistinguishability through a series of games.
SimTally algorithm. The SimTally(BB, r) algorithm performs the Tally procedure on BB, except that 1) all the proofs are simulated and 2) in steps 14 and 17 of the Tally procedure, the decryption results are based on r.

Now, we prove the indistinguishability through a sequence of games. We start with the adversary interacting with the challenger with $b = 0$ and end up with the adversary interacting with the challenger with $b = 1$.
Game G_0**.** Let G_0 be the experiment $\text{Exp}^{BP}_{\mathcal{A},\mathcal{V}}[0](\lambda, C)$ (see Figure 8 and Definition 1).
Game G_1**.** Game G_1 is the same as G_0, except that all proofs in the Tally procedure are simulated.
Claim 1: Because of the zero-knowledge property of the proofs, G_1 and G_0 are indistinguishable.
Game G_2**.** Game G_2 is the same as G_1, except that $\mathcal{O}$board returns BB_1 instead of BB_0.
Claim 2: If the ElGamal encryption scheme is IND-CPA secure and the NIZK $\text{NIZK}_{\text{knowledge}}$ is complete, simulation-sound extractable, and zero-knowledge, then G_2 and G_1 are indistinguishable.
Proof: Let n be the number of $\mathcal{O}$voteLR calls made by the adversary $\mathcal{A}$. We now build a series of games $H_0, \ldots, H_n$ and prove indistinguishability by a hybrid argument. Game H_i tracks a bulletin board BB. When the adversary calls $\mathcal{O}\text{cast}(\beta)$, the challenger performs $\text{BB} \leftarrow \text{BB}||\beta$; for the first i calls to $\mathcal{O}$voteLR, the challenger performs $\text{BB} \leftarrow \text{BB}||\beta_1$; for the remaining calls, the challenger performs $\text{BB} \leftarrow \text{BB}||\beta_0$. Note that $H_0 = G_1$ and $H_n = G_2$.

As proved by Bernhard *et al.* [12], an IND-CPA secure encryption with simulation-sound extractable ZKP is NM-CPA (non-melleable) secure. The NM-CPA experiment is depicted in Figure 17. The adversary may only call the oracle $\mathcal{O}$dec once. The adversary wins if $b' = b$ and his advantage is defined as $|\Pr[b' = b] - 1/2|$.

Now, we show that if an adversary $\mathcal{A}_i$ can distinguish H_i from H_{i-1}, we can construct adversary $\mathcal{B}_i$ that breaks the NM-CPA property of the encryption scheme. Adversary $\mathcal{B}_i$ receives the public key pk from its challenger. At the start of the game, $\mathcal{B}_i$ performs the Setup procedure and sets $\text{pk}_\text{T} = \text{pk}$. It answers the jth $\mathcal{O}$voteLR query as follows:

- If $j < i$, it sets $\text{BB} \leftarrow \text{BB}||\beta_1$.
- If $j = i$, it sends v_0, v_1 to the NM-CPA challenger and receives a challenge ciphertext c^*. It uses c^* to obtain a ballot β^* and sets $\text{BB} \leftarrow \text{BB}||\beta^*$.
- If $j > i$, it sets $\text{BB} \leftarrow \text{BB}||\beta_0$.

```
$\underline{\mathrm{Exp}^{\text{NM-CPA}}_{\mathcal{A}}(\lambda):}$
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Keygen}(1^\lambda)$;
    $m_0, m_1 \leftarrow \mathcal{A}(\mathsf{pk})$;
    $b \leftarrow_\$ \{0,1\}$;
    $c^* \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}\mathrm{dec}}(\mathsf{pk}, c^*)$;
$\mathcal{O}\mathrm{dec}(\boldsymbol{c})$ :
    if $c^* \in \boldsymbol{c}$ then return $\perp$;
    for each $i$, $m_i \leftarrow \mathrm{Dec}(\mathsf{sk}, c_i)$;
    return $\boldsymbol{m} = \{m_i\}_i$.
```

Figure 17: NM-CPA Experiment $\mathrm{Exp}^{\text{NM-CPA}}_{\mathcal{A}}(\lambda)$

When the adversary $\mathcal{A}_i$ calls $\mathcal{O}$tally, $\mathcal{B}_i$ proceeds as follows. Let $\Gamma := (\boldsymbol{u}_{i_1}, \pi_{i_1}), \ldots, c^* = (\boldsymbol{u}_j, \pi_j), \ldots, (\boldsymbol{u}_{i_k}, \pi_{i_k})$ be the vote ciphertexts and proofs in the valid ballots. $\mathcal{B}_i$ queries $\mathcal{O}$dec using $\Gamma \backslash c^*$ and obtains $(v_{i_1}, \ldots, v_{j-1}, v_{j+1}, \ldots, v_{i_k}) = \mathcal{O}\mathrm{dec}(\Gamma \backslash c^*)$. It sets the decryption result as $(v_{i_1}, \ldots, v_{j-1}, v_0, v_{j+1}, \ldots, v_{i_k})$ and continues the Tally procedure by simulating the proofs.

We can see that if $b = 0$ in $\mathcal{B}_i$'s NM-CPA game, then $\mathcal{B}_i$ perfectly simulates H_{i-1}; if $b = 1$ in $\mathcal{B}_i$'s NM-CPA game, then $\mathcal{B}_i$ perfectly simulates H_i. Thus, $\mathcal{B}_i$ breaks NM-CPA games if $\mathcal{A}_i$ distinguishes H_i from H_{i-1}. By a standard hybrid argument, $H_0 = G_1$ is indistinguishable from $H_n = G_2$.

Observe that the adversary's view in G_2 is identical to the view in the experiment $\mathrm{Exp}^{BP}_{\mathcal{A},\mathcal{V}}[1](\lambda, C)$. This concludes the proof. □

C.2 Strong consistency and strong correctness

Here, we show that our scheme meets strong consistency and strong correctness as defined by Bernhard *et al.* [10]. Intuitively, strong consistency ensures that the result r is equal to the result function applied directly to the valid ballots. Strong correctness ensures that no adversary can generate a bulletin board BB such that $\mathsf{Validate}(\mathsf{BB}, \beta) = \perp$ for an honestly generated ballot β. The strong consistency experiment and strong correctness experiment are depicted in Figure 18 and Figure 19, respectively.

```
$\underline{\mathrm{Exp}^{\text{s-cons}}_{\mathcal{A},\mathcal{V}}(\lambda, C):}$
    $((\mathsf{pk}_{\mathsf{RA}}, \mathsf{sk}_{\mathsf{RA}}), (\mathsf{pk}_{\mathsf{T}}, \mathsf{sk}_{\mathsf{T}})) \leftarrow \mathsf{Setup}(1^\lambda, C)$;
    $\mathsf{BB} \leftarrow \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}, \mathsf{pk}_{\mathsf{T}})$;
    $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathsf{sk}_{\mathsf{T}})$;
    if $r \neq \rho(\mathsf{Extract}(\mathsf{sk}_{\mathsf{T}}, \beta_1), \ldots, \mathsf{Extract}(\mathsf{sk}_{\mathsf{T}}, \beta_n))$
    then return $\top$ else return $\perp$.
```

Figure 18: Strong Consistency Experiment $\mathrm{Exp}^{\text{s-cons}}_{\mathcal{A},\mathcal{V}}(\lambda, C)$

Definition 4. *Strong consistency. Let* $\mathcal{V} = \{\mathsf{Setup}, \mathsf{Reg}, \mathsf{FakeReg}, \mathsf{E\_reg}, \mathsf{V\_vote}, \mathsf{E\_vote}, \mathsf{Validate}, \mathsf{Tally}, \mathsf{VerifyElection}\}$ *be an election scheme for a candidate list* C*, security parameter* λ*, and result function* ρ*. We say that* $\mathcal{V}$ *meets strong consistency if there exists functions* Extract *and* ValidInd *that satisfy the following conditions:*

(1) *For* $((\mathsf{pk}_{\mathsf{RA}}, \mathsf{sk}_{\mathsf{RA}}), (\mathsf{pk}_{\mathsf{T}}, \mathsf{sk}_{\mathsf{T}})) \leftarrow \mathsf{Setup}(1^\lambda, C)$*, for all* $(\mathsf{pk}, \mathsf{sk})$ *output by* $\mathsf{Reg}(id)$*, and for any ballot* $\beta \leftarrow \mathsf{V\_vote}(\mathsf{pk}, \mathsf{sk}, v)$*, we have* $\mathsf{Extract}(\mathsf{sk}_{\mathsf{T}}, \beta) = v$.
(2) *For any bulletin board and ballot generated by any PPT adversary* $\mathcal{A}$ *such that* $(\mathsf{BB}, \beta) \leftarrow \mathcal{A}$ *and* $\mathsf{Validate}(\mathsf{BB}, \beta) = \top$*, then* $\mathsf{ValidInd}(\beta) = \top$.
(3) *For any PPT adversary* $\mathcal{A}$*, the advantage* $\Pr[\mathrm{Exp}^{\text{s-cons}}_{\mathcal{A},\mathcal{V}}(\lambda, C) = \top]$ *is a negligible function in the security parameter* λ.

```
$\underline{\mathrm{Exp}^{\text{s-corr}}_{\mathcal{A},\mathcal{V}}(\lambda, C):}$
    $((\mathsf{pk}_{\mathsf{RA}}, \mathsf{sk}_{\mathsf{RA}}), (\mathsf{pk}_{\mathsf{T}}, \mathsf{sk}_{\mathsf{T}})) \leftarrow \mathsf{Setup}(1^\lambda, C)$;
    $(id, v, \mathsf{BB}) \leftarrow \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}, \mathsf{pk}_{\mathsf{T}})$;
    $\beta \leftarrow \mathsf{V\_vote}(\mathsf{pk}_{id}, \mathsf{sk}_{id}, v)$;
    if $\mathsf{Validate}(\mathsf{BB}, \beta) = \perp$ then return $\top$ else return $\perp$.
```

Figure 19: Strong Correctness Experiment $\mathrm{Exp}^{\text{s-corr}}_{\mathcal{A},\mathcal{V}}(\lambda, C)$

Definition 5. *Strong correctness. Let* $\mathcal{V} = \{\mathsf{Setup}, \mathsf{Reg}, \mathsf{FakeReg}, \mathsf{E\_reg}, \mathsf{V\_vote}, \mathsf{E\_vote}, \mathsf{Validate}, \mathsf{Tally}, \mathsf{VerifyElection}\}$ *be an election scheme for a candidate list* C*, and security parameter* λ*. We say that* $\mathcal{V}$ *meets strong correctness if for any PPT adversary* $\mathcal{A}$*:*

$$\Pr[\mathrm{Exp}^{\text{s-corr}}_{\mathcal{A},\mathcal{V}}(\lambda, C) = \top]$$

is a negligible function in the security parameter λ.

Theorem 4. *Under the same assumptions as theorem 1, the voting protocol described in section 4 provides strong consistency and strong correctness.*

Proof. We first define the functions Extract and ValidInd as follows:

(1) $\mathsf{Extract}(\mathsf{sk}_{\mathsf{T}}, \beta)$ takes as input the extraction key sk_{T} and the ballot $\beta = (\mathsf{pk}, \boldsymbol{u}, \pi, \sigma)$. It checks the proof and signature in β. It returns $\perp$ if any of these checks fail; otherwise, it computes $v \leftarrow \mathsf{EC.Dec}_{\mathsf{sk}_{\mathsf{T}}}(\boldsymbol{u})$ and returns (pk, v).
(2) $\mathsf{ValidInd}(\beta)$ checks the proof and signature in β. It returns $\top$ if and only if the checks pass.

The first condition of strong consistency is satisfied by the completeness of the zero-knowledge proofs, the correctness of signatures, and the correctness of ElGamal encryption scheme.

The second condition of strong consistency is satisfied because ValidInd executes a strict subset of the checks in Validate.

For the third condition, by the correctness of ElGamal decryption and the correctness of shuffle, the election results output by the Tally algorithm and the result function ρ are identical.

To prove strong correctness, we observe that an honestly generated ballot is not appended to the bulletin board if the corresponding pk already appears in another ballot. By the EUF-CMA property of the signature scheme, the adversary $\mathcal{A}$ has negligible probability of generating a valid signature under pk without knowing sk, so $\mathcal{A}$'s advantage in $\mathrm{Exp}^{\text{s-corr}}_{\mathcal{A},\mathcal{V}}(\lambda, C)$ is negligible. □

D PROOF OF VERIFIABILITY

Proof of theorem 2. Let the adversary $\mathcal{A}$ output a set of registration items, a set of votes, the result r, and the tally proof Π. Let $T = \{\beta_1, \ldots, \beta_n\}$ be the valid ballots on the BB. By the homomorphism of ElGamal encryption scheme and the soundness of the tally proofs, we can conclude that r is the correct tally of $T = \{\beta_1, \ldots, \beta_n\}$ if $\mathsf{VerifyElection}(\mathsf{BB}, (r, \Pi))$ returns $\top$.

Now, we prove that for each ballot $\beta \in T$, it is either not counted or the re-randomized ballot of one of the following sets:

- $H = \{(\mathsf{pk}_i^h, v_i^h, \alpha_i^h, *)\}_{i=1}^{n_h}$, the votes of the honest voters who have checked their ballots.
- $H' = \{(\mathsf{pk}_i^{h'}, v_i^{h'}, \alpha_i^{h'}, *)\}_{i=1}^{n_{h'}}$, the votes of the honest voters who have not checked their ballots.
- $\mathsf{Corrupted} = \{\mathsf{pk}_i^c, v_i^c, \alpha_i^c\}_{i=1}^{n_c}$, the votes of the corrupted voters.

The soundness of $\mathsf{NIZK}_{\mathsf{power}}$ and $\mathsf{NIZK}_{\mathsf{DVP\text{-}reenc}}$ ensures that the registration items are formed correctly. The EUF-CMA property of the signature scheme ensures that the adversary cannot create a valid signature for an honest voter. Therefore, for each valid ballot $\beta \in T$, if it does not correspond to the above three sets, either it will be dropped or will match a fake registration item in the Tally procedure; in both cases, it will not be counted.

We now prove that the adversary cannot remove the votes of the honest voters who have checked their ballots. By the soundness of $\mathsf{NIZK}_{\mathsf{sk}}$, the adversary cannot create a valid "fake registration item" for pk if he does not know the corresponding sk. Thus, all the valid ballots will match the corresponding real registration item and be counted in the tally phase.

Finally, if a corrupted voter generates a ballot that encrypts an invalid candidate, it will be dropped in the tally phase. We can conclude that if the result r verifies, then it must correspond to the result of the tally function ρ computed on all the votes by honest voters who checked their ballots, at most n_c votes cast by corrupted voters, and a subset of votes cast by honest voters who did not check their ballots. □

E PROOF OF COERCION-RESISTANCE

Proof of theorem 3. To prove this theorem, we first construct SimTally and SimView algorithms and prove indistinguishability through a series of games.

SimTally algorithm. The $\mathsf{SimTally}(\mathsf{BB}, r)$ algorithm is the same as the one we define in the proof of theorem 1. It performs the Tally procedure on BB, except that 1) all the proofs are simulated and 2) in steps 14 and 17 of the Tally procedure, the decryption results are based on r.

SimView algorithm. The $\mathsf{SimView}(\mathsf{sk}_{id})$ performs as follows. Let $\langle K, A, \mathsf{tx}, \delta\rangle$ be the registration message and $\langle \tilde{K}, A, \mathsf{tx}, \delta, \sigma_{\mathsf{RA}}\rangle$ be the real registration item on BB. It computes $K'' \leftarrow \mathsf{EC.Enc}_{\mathsf{pk_T}}(\mathsf{pk}')$ and claims that $\tilde{K}$ is re-encryption of K'' by simulating the designated verifier proof $\mathsf{NIZK}_{\mathsf{DVP\text{-}reenc}}$, which requires sk_{id}. It returns the simulated view by replacing K with K'' and the real proof with the simulated proof.

Now, we prove the indistinguishability through a sequence of games. We start with the adversary interacting with the challenger with $b = 0$ and end up with the adversary interacting with the challenger with $b = 1$.

Game G_0. Let G_0 be the experiment $\mathrm{Exp}^{CR}_{\mathcal{A},\mathcal{V}}[0](\lambda, C)$ (see Figure 10 and Definition 3).

Game G_1. Game G_1 is the same as G_0, except that all proofs in the Tally procedure are simulated.

Claim 1: Because of the zero-knowledge property of the proofs, G_1 and G_0 are indistinguishable.

Game G_2. Game G_2 is the same as G_1, except that 1) $\mathcal{O}$receipt returns β_1 and the simulated view, and 2) $\mathcal{O}$board returns BB_1 instead of BB_0.

Claim 2: If the ElGamal encryption scheme is IND-CPA secure, the NIZKs NIZK_i, $i \in \{\mathsf{knowledge}, \mathsf{DVP\text{-}reenc}, \mathsf{sk}\}$ are complete, sound, and zero-knowledge, and $\mathsf{NIZK}_{\mathsf{sk}}$, $\mathsf{NIZK}_{\mathsf{knowledge}}$ are simulation-sound extractable, then G_2 and G_1 are indistinguishable.

Proof: First, the adversary cannot distinguish the real view and the simulated view in the registration phase because of the zero-knowledge property of the designated verifier proof.

Second, by the simulation-sound extractability of $\mathsf{NIZK}_{\mathsf{sk}}$, the adversary cannot duplicate a "fake registration item" by re-randomizing an existing one. Thus, he cannot distinguish a fake ballot from a real ballot by the Tally procedure.

Now, except for seeing the simulated view instead of the real view, the only difference between G_2 and G_1 is that the adversary sees different encrypted ballots. Then, the rest of the proof is very similar to the proof of theorem 1. We define a sequence of games that replace the ballots on BB_0 with the ballots on BB_1 one by one. By a standard hybrid argument, the indistinguishability between G_2 and G_1 is reduced to the NM-CPA property of the underlying encryption scheme.

Finally, we observe that the adversary's view in G_2 is identical to the view in the experiment $\mathrm{Exp}^{CR}_{\mathcal{A},\mathcal{V}}[1](\lambda, C)$. This concludes the proof. □