VeriVoting: A decentralized, verifiable and privacy-preserving scheme for weighted voting

Xiaohan Yue

Shenyang University of Tech.

xhyue@sut.edu.cn

ABSTRACT

Decentralization, verifiability, and privacy-preserving are three fundamental properties of modern e-voting. In this paper, we conduct extensive investigations into them and present a novel e-voting scheme, VeriVoting, which is the first to satisfy these properties. More specifically, decentralization is realized through blockchain technology and the distribution of decryption power among competing entities, such as candidates. Furthermore, verifiability is satisfied when the public verifies the ballots and decryption keys. And finally, bidirectional unlinkability is achieved to help preserve privacy by decoupling voter identity from ballot content. Following the ideas above, we first leverage linear homomorphic encryption schemes and non-interactive zero-knowledge argument systems to construct a voting primitive, SemiVoting, which meets decentralization, decryption-key verifiability, and ballot privacy. To further achieve ballot ciphertext verifiability and anonymity, we extend this primitive with blockchain and verifiable computation to finally arrive at VeriVoting. Through security analysis and performance evaluations, VeriVoting offers a new trade-off between security and efficiency that differs from all previous e-voting schemes and provides a radically novel practical approach to large-scale elections.

CCS CONCEPTS

•Security and privacy~Cryptography •Security and privacy ~Security services~Privacy-preserving protocols •Security and privacy~Security services~Pseudonymity, anonymity and untraceability

KEYWORDS

Decentralization, Verifiability, Privacy-preserving, Weighted Voting

1 INTRODUCTION

Electronic voting (e-voting), an election method for people to freely express their will, plays a crucial role in the process of democratization and development with its efficiency and flexibility. In all e-voting schemes, it is essential to ensure that the final tally result correctly reflects the ballots cast by the voters. Moreover, voters' ballots must remain private so that the final result is not biased by those afraid to express their will freely. Another significant challenge in centralized e-voting systems is the issue of trust, which means that the correctness of the tally result as well as the privacy of ballots and voters must be guaranteed by one or more trusted entities. However, finding such entities that everyone trusts in real life is not easy. The above basic security and privacy requirements in modern e-voting systems can be further explained as the following properties.

Decentralization. In decentralized e-voting schemes, the process and outcome of voting cannot be controlled by a single party. As a consequence, existing decentralized voting schemes commonly use cryptographic technologies, such as ring signatures[1,2], secret sharing[3], multi-party computation[4,5], blockchain[6,7], etc., to distribute trust to multi-entities. However, these decentralized schemes are typically designed for small-scale elections focusing on security with minimal trust assumptions[8]. Two issues limit the scalability: One is performance. The cost of the voting or tally algorithm running on the voter's side is linearly related to the number of participating voters, such as [1],[4],[5]; Another is dynamic join. Due to some system parameters (such as public parameters, decryption keys, etc.) are determined by a fixed number of voters before the voting phase, most decentralized e-voting schemes do not support dynamic join for voters.

Verifiability. In decentralized voting systems, the verifiability of the process and outcome of voting is particularly crucial due to the lack of trust[9]. Verifiability for the decentralized voting system should consist of the following aspects: The first is ballot verifiability, which prevents dishonest voters from voting without following the prescribed rules; The second is decryption-key verifiability, which is optionally considered due to the difference in e-voting construction, can prevent decryption-key holders from submitting false ones to sway the outcome of elections; The third is tally result verifiability, which enables external and internal entities to detect and reject false election results.

Privacy-preserving. Privacy-preserving on ballots and voters are called ballot privacy[10] and anonymity[11], respectively. Ballot privacy means that no coalition of malicious parties (consisting of partial voters or decryption key holders) can learn an honest voter's ballot content. Anonymity enables voters to cast their ballots anonymously, and even if the ballot content was revealed, individual ballots could not be traced back to individual voters. Essentially, ballot privacy and anonymity protect the content of ballots and voters' identities and further imply bidirectional unlinkability between ballots and voters.

Our roadmap. In a weighted voting scheme, the tallying algorithm for a candidate $\mathbb{c}_j$ can be mathematically expressed as an inner operation $t_j =< \vec{v}_j, \vec{w} >$, where t_j is the final score for the candidate $\mathbb{c}_j$, $\vec{v}_j$ is the vector consisting of ballots cast by voters for the candidate $\mathbb{c}_j$, and $\vec{w}$ is the vector consisting of corresponding voters' weights. As such, from a conceptual standpoint, decentralized variants of functional encryption(FE) schemes with ciphertext indistinguishability (IND)-security, such as DMCFE[12], DDFE[13], and ad hoc MIFE[14], are good candidates to meet not only the functionality requirements for decentralized e-voting but also the security requirements for ballot privacy.

However, as described in the definition of DDFE[13], the output of functions, including the inner product function, is determined by participants' inputs and decryption keys. Thus, once some participants are offline or absent after the encryption operation, the decryption algorithm cannot work to output the result of functions because the complete decryption key cannot be aggregated from all participants' partial decryption keys. Alternatively, if a malicious participant provides a false decryption key, then the output of the decryption algorithm would be incorrect. To fill the gap of lacking verifiable, in 2023, Nguyen et al.[15] presents verifiable decentralized inner product FE. However, generating a partial decryption key relies on multi-party protocols, which makes verifying each key highly inefficient. Another drawback of this solution[15] is that a receiver cannot detect which participant sent a malicious key share if the aggregated decryption key is invalid. Inspired by[13],[15], the verifiability of decryption (i.e., tally result) is conducted by the verifiability of encryption and decryption key in our scheme so as to get rid of the performance and centralization problems caused by the existing verifiable e-voting systems which focus on the verifiability of encryption and decryption.

SemiVoting. The starting point of our paper is to design a decentralized weighted voting primitive for semi-honest voters, named SemiVoting, which features ballot privacy and efficient decryption-key verifiability.

Our primitive of SemiVoting is based on two facts about large-scale elections: **Fact 1**. The candidates will not be absent throughout the election, and the size of candidates is always smaller than the size of voters, even if candidates are also voters. **Fact 2**. As fellow competitors, the candidates never collude with each other, or at least one of the candidates does not collude with the other candidates. These two facts make candidates being holders of encryption keys and decryption keys in our construction such that: In the voting phase, without generating the encryption and decryption keys, each voter can dynamically participate in voting activities and encrypt their ballots using aggregated encryption keys of candidates; In the decryption-key generation phase, each candidate generates verifiable partial decryption keys, which can be aggregated into one decryption key by anyone after verifying the validity of all the partial decryption keys; In the tally phase, anyone can decrypt aggregated encrypted ballots using the decryption keys. In each phase, the number of keys and the cost of algorithms are related to the size of candidates, not voters, which enables our construction to be feasible for large-scale elections. The detail of the SemiVoting primitive is shown in Section 2.

SemiVoting, however, only accomplishes half of the work in our scheme, namely decentralization, decryption-key verifiability, and ballot privacy. For dishonest voters, they could not follow the rule of voting, such as duplicate voting[16,17], casting ill-formed ballots [18], etc. Thus, from the perspective of object-oriented programming, we derive a new class named VeriVoting from the SemiVoting primitive to fill the gap that prevents this primitive from being practical.

VeriVoting. Similar to other decentralized private computation schemes, such as Zcash[19], Hawk[20], and Zexe[21], we mainly add two extra components for the voting transaction, including a commitment scheme and a verifiable computation system, i.e., zkSNARKs[22-24], to avoid voter's dishonest behaviors and achieve encrypted ballot verifiability while keeping voters' anonymity. At a high level, as a result of horizontal extension on the base class, we decouple ballot privacy and anonymity by using different privacy components where the linear homomorphic encryption scheme is for ballot privacy, and the zkSNARK system is for anonymity. The detail of VeriVoting is shown in Section 3.

Our contributions. Following the above roadmap, our contributions are summarized below.

(1). We first introduce a novel weighted voting primitive, SemiVoting, which features decentralization, decryption-key verifiability, and ballot privacy. In addition, the computational complexity of the voting and tally algorithms is only related to the number of candidates rather than voters. Furthermore, based on bounded-collusion indistinguishability secure encryption schemes with linear homomorphic properties and non-interactive zero-knowledge (NIZK) arguments system, we give a feasible general construction later used by VeriVoting.

(2). To fill the gap in making SemiVoting practical, we employ zkSNARKs and blockchain technologies to build our final weighted voting scheme, VeriVoting. Benefiting from both technologies, VeriVoting provides ballot verifiability and voter anonymity and realizes the horizontal extension of SemiVoting.

(3). With the help of the Remix IDE[25] and Zokrates toolbox [26], we evaluate the performance of core algorithms in VeriVoting on the smart contract side and the voter and candidate side, respectively. The result shows that the computation cost of the algorithms is related to the number of candidates, not voters, which makes VeriVoting suitable for large-scale elections.

(4). With the linear homomorphism of ballot ciphertexts, we present a distributed framework to solve the problem that a single contract cannot be executed in parallel for large-scale elections.

Table 1: Symbols

Symbols	Meaning
$\mathbb{V}$	Set of valid voters
$\mathbb{v}_i$	Voter i, $\mathbb{v}_i \in \mathbb{V}$
$\mathbb{a}$	Aggregator
$\mathbb{C}$	Set of candidates
$\mathbb{c}_j$	Candidate j, $\mathbb{c}_j \in \mathbb{C}$
λ	Security parameter
pk_j, sk_j	Candidate $\mathbb{c}_j$'s public and secret key
dk_j	Decryption key generated by $\mathbb{c}_j$'s secret key sk_j
$\vec{v}_i$	Voter $\mathbb{v}_i$'s ballot vector, $\vec{v}_i \in \mathcal{B}^{\lvert\mathbb{C}\rvert}$
$\vec{v}_j$	Vector of ballots casting for the candidate $\mathbb{c}_j$, $\vec{v}_j \in \mathcal{B}^{\lvert\mathbb{V}\rvert}$
$\vec{w}$	Valid voters' weight vector, $\lvert\vec{w}\rvert = \lvert\mathbb{V}\rvert$
$\overrightarrow{ct}_i$	Encrypted/Secret ballot vector, $\overrightarrow{ct}_i \in \mathcal{C}^{\lvert\mathbb{C}\rvert}$
$\mathbb{CT}$	Iteratively aggregatable ciphertext set, $\lvert\mathbb{CT}\rvert = \lvert\mathbb{C}\rvert$.
$\mathbb{DK}$	Iteratively aggregatable decryption-key set, $\lvert\mathbb{DK}\rvert = \lvert\mathbb{C}\rvert$

2 SEMIVOTING PRIMITIVE

In this section, we introduce a weighted voting construction for semi-honest voters in which the computational cost of the voting and tally algorithm is related to the number of candidates rather than voters. Moreover, since the number of voters is always smaller than the candidates in most voting applications, we call our scheme *moderately succinct*. To show our detailed construction clearly, we will first introduce the overview of the SemiVoting primitive, mainly including introductions of entities and a formal definition of SemiVoting. Then, we give informal descriptions of security requirements that our scheme should guarantee.

The parameters of the primitive are described in Table 1.

2.1 Overview

The overview of SemiVoting is shown in Figure 1, which consists of three entities, namely voters, candidates, and an aggregator, and four phases, including initialization, voting, decryption-key (DK) generation, and tally phases. Notes that we omit the registration process in this section, which we include in VeriVoting.

Voter. Each Voter $\mathbb{v}_i \in \mathbb{V}$ is assumed semi-honest, which means the behaviors of each voter follow the scheme's rules, but they are curious about other voters' ballot content.

Candidate. Each candidate $\mathbb{c}_j \in \mathbb{C}$ is malicious, which means they will break the fairness of the voting via some illegal methods, such as collusion, submitting false decryption keys, etc.

Aggregator. The aggregator $\mathbb{a}$ initializes the bulletin board, and adds $\mathbb{CT}$ and $\mathbb{DK}$ to the bulletin board. The internal state of the aggregator $\mathbb{a}$ is public, which means all operation processes and state variables inside the aggregator are public to all parties. (In VeriVoting, we replace the aggregator with a smart contract.).

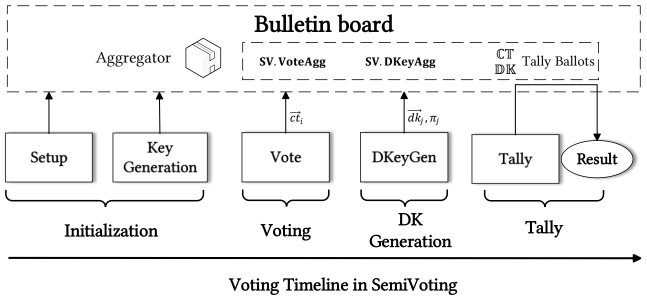

Figure 1: Overview of SemiVoting

Definition 1 (SemiVoting, SV). A SemiVoting primitive on ciphertext space $\mathcal{C}$, decryption-key space $\mathcal{K}$ and ballot space $\mathcal{B}$, over a set $\mathbb{V}$ of voters, a set $\mathbb{C}$ of candidates and an aggregator $\mathbb{a}$, is defined by the following algorithms:

- $pp_{sv} \leftarrow \mathsf{Setup}\left(1^{\lambda}\right)$: Takes as input a security parameter λ, and generates public parameters pp_{sv}.
- $(pk_j, sk_j) \leftarrow \mathsf{KeyGen}(pp_{sv})$: Each candidate $\mathbb{c}_j \in \mathbb{C}$ takes as input the public parameters pp_{sv} and runs the algorithm, outputs his/her public-private keys (pk_j, sk_j).
- $\overrightarrow{ct_i} \leftarrow \mathsf{Vote}\left(pp_{sv}, \{pk_j\}_{j\in[\mathbb{C}]}, \vec{v}_i, w_i\right)$: Each Voter $\mathbb{v}_i \in \mathbb{V}$ takes as input the public parameter pp_{sv}, public keys $\{pk_j\}_{j\in[\mathbb{C}]}$, a ballot vector $\vec{v}_i \in \mathcal{B}^{|\mathbb{C}|}$ and her weight w_i, outputs ciphertexts $\overrightarrow{ct_i} \in \mathcal{C}^{|\mathbb{C}|}$.
  - $\perp/\mathbb{CT} \leftarrow \mathsf{VoteAgg}(\mathbb{CT}', \overrightarrow{ct_i})$: Upon receiving the ballot ciphertexts $\overrightarrow{ct_i}$ from a voter $\mathbb{v}_i$, the aggregator $\mathbb{a}$ iteratively updates its state variables set $\mathbb{CT}$ by aggregating $\overrightarrow{ct_i}$ into the last state set $\mathbb{CT}'$.
- $\left(\overrightarrow{dk_j}, \pi_j\right) \leftarrow \mathsf{DKeyGen}(sk_j, \mathbb{CT})$: When the voting phase finished, each candidate $\mathbb{c}_j \in \mathbb{C}$ takes the aggregated set $\mathbb{CT}$ and secret key sk_j as inputs, outputs the vector consisting of partial decryption keys $\overrightarrow{dk_j} \in \mathcal{K}^{|\mathbb{C}|}$ and a proof π_j to the aggregator $\mathbb{a}$.
  - $\perp/\mathbb{DK} \leftarrow \mathsf{DKeyAgg}(\overrightarrow{dk_j}, \pi_j, \mathbb{DK}')$: Takes as input the partial decryption-key vector $\overrightarrow{dk_j}$ from a candidate $\mathbb{c}_j$ and corresponding proof π_j, the aggregator $\mathbb{a}$ first checks the validity of partial decryption-keys in $\overrightarrow{dk_j}$ with the proof π_j, if all is valid, updates its state variable set $\mathbb{DK}$ by aggregating $\overrightarrow{dk_j}$ into $\mathbb{DK}'$; Otherwise, outputs $\perp$ and abort.
- $\{t_j\}_{j\in[\mathbb{C}]} \leftarrow \mathsf{Tally}(\mathbb{DK}, \mathbb{CT})$: Takes as input the aggregated ciphertext set $\mathbb{CT}$ and the decryption-key set $\mathbb{DK}$, outputs final weighted tallies $\{t_j =< \vec{v}_j, \vec{w} >\}_{j\in[\mathbb{C}]}$.

2.2 Security Requirements

SemiVoting should follow the below security requirements.

Decentralization. Similar to decentralized functional encryptions, our SemiVoting primitive allows aggregating data coming from different parties and does not require a trusted party with a master secret key, such that no centralized entity or organization can change or control the outcome of elections.

Ballot Privacy. It means that, in the event of collusion among at most $\delta_c \coloneqq (|\mathbb{C}| - 1)$ candidates or $\delta_v \coloneqq (|\mathbb{V}| - |\mathbb{V}_h|)$ voters, the contents of ballots of the target voters in the honest-voter set $\mathbb{V}_h$ cannot be revealed. In addition, we assume that the size of set $\mathbb{V}_h$ is at least equal to two, and the voters in set $\mathbb{V}_h$ can not cast their ballots for the same candidate.

Decryption-Key Verifiability. According to the definition of functional encryption[13], the output of the encryption scheme is determined by inputs consisting of decryption keys and messages. Thus, a malicious candidate can easily generate a false decryption key to affect the election outcome. To avoid cheating, all decryption keys generated by candidates should be verifiable.

2.3 Building Blocks

This section briefly introduces related technologies, including BC-IND-secure encryption schemes and NIZK argument system, which will be used as the building blocks for one feasible construction of SemiVoting in the following section. Moreover, we illustrate the required properties related to these building blocks.

(1) Bounded-collusion indistinguishability(BC-IND)-secure encryption schemes

In our construction, the ciphertext $ct = \mathcal{E}.\mathsf{Encrypt}(pk, x; r)$ needs to consist of two parts ct^0 and ct^1. The first part $ct^0 = \mathsf{Comm}(r)$ is a commitment to the randomness r used for the encryption. The second part ct^1 is the encryption $\mathsf{Enc}(pk, x; r)$ of the plaintext x under public key pk and randomness r. Accordingly, the decryption algorithm $\mathcal{E}.\mathsf{Decrypt}(sk, ct)$ consist of two routines $\mathsf{KeyGen}(sk, ct^0) \rightarrow dk$ and $\mathsf{Dec}(dk, ct^1) \rightarrow x$. The BC-IND-secure encryption needs to meet the following properties:

Linear Commitment Homomorphism. We say that a commitment scheme has linear commitment homomorphism (LCH, for short) if $ct^0 \leftarrow \mathsf{Comm}(r_1 + r_2)$ can be efficiently computed from $ct_1^0 \leftarrow \mathsf{Comm}(r_1)$ and $ct_2^0 \leftarrow \mathsf{Comm}(r_2)$.

Linear Encryption Homomorphism. We say that public key encryption has linear encryption homomorphism (LEH, for short) if $ct^1 \leftarrow \mathsf{Enc}(pk_1 \cdot pk_2, x_1 + x_2; r_1 + r_2)$ can be efficiently computed from $ct_1^1 \leftarrow \mathsf{Enc}(pk_1, x_1; r_1)$ and $ct_2^1 \leftarrow \mathsf{Enc}(pk_2, x_2; r_2)$.

Linear Key Homomorphism. We say that public key encryption has linear key homomorphism (LKH, for short) if the decryption

$\mathcal{E}.\mathrm{Decrypt}(sk_1 + sk_2, ct) \to \mathrm{Dec}(dk_1 \cdot dk_2, ct^1) \to \sum x_i$ can correctly recover the aggregated plaintext from decryption keys $\left(dk_1 \leftarrow \mathrm{KeyGen}(sk_1, ct^0), dk_2 \leftarrow \mathrm{KeyGen}(sk_2, ct^0)\right)$, where $ct \coloneqq (ct^0, ct^1)$ is the aggregated ciphertext by LEH and LCH properties.

Discussion. By the LKH property, the decryption keys from all candidates can be combined to decrypt the aggregated ciphertext in the tally phase. However, our construction is no longer secure if all candidates collude to decrypt a target ciphertext. Therefore, based on **Fact 2**, we need encryption schemes to be secure against a bounded number of collusions, i.e., at most δ_c candidates. This security property is slightly similar to bounded-collusion(BC) in IBE[121]-[123] or called key-insulated[27] in PKE. The difference is that in BC-IBE, n public-secret key pairs are held by a single entity, whereas in our scheme, n public-secret key pairs are held separately by n candidates. Thus, the essential difference about bounded-collusion is that in BC-IBE schemes, the attacker who owns $t-1$ linear combination secret keys cannot create t'th decryption key for $id_t \in \mathcal{ID}$ which is guaranteed by a cover-free map $\phi: \mathcal{ID} \to \{0,1\}^n$, while in SemiVoting, the linear combination of secret keys is completed under linear homomorphism one-way functions[28], e.g., $f_{ow}(a; b) \coloneqq (g^b)^a$ under the computational Diffie-Hellman (CDH) assumption.

Therefore, our security against bounded-collusion is straightforward, that is, if homomorphic one-way functions exist, even though δ_c malicious candidates collude to compute $\{f_{ow}(sk^*; r_k)\}_{k\in[|\mathbb{C}|]}$ from $\left\{f_{ow}\left(\sum_j^{[\mathbb{C}]} sk_j\, ; r_k\right)\right\}_{k\in[|\mathbb{C}|]}$ with their secret keys $\{sk_j\}_{j\in[\mathbb{C}\backslash \mathbb{c}^*]}$, and given $f_{ow}(sk^*; 1)$ and $f_{ow}(1; r_t)$, they cannot compute $f_{ow}(sk^*; r_t)$ without knowing the target candidate $\mathbb{c}^*$'s secret key sk^*. Obviously, based on the CDH assumption, if $f_{ow}(sk; r) \coloneqq (g^r)^{sk}$, the security against bounded-collusion holds. In Lemma A.4, we prove the property through an instance.

(2). Non-interactive zero-knowledge argument systems

For the verifiability of decryption keys, we employ non-interactive zero-knowledge(NIZK) argument systems[29-31] to convince the verifier that the public key and decryption key are generated using the same secret key. A NIZK argument system Π_{NIZK} consists of two core algorithms: $\Pi_{\mathrm{NIZK}}.\mathrm{Prove}(crs, s, \omega) \to \pi$, $\Pi_{\mathrm{NIZK}}.\mathrm{Verify}(crs, s, \pi) \to 1/0$, where $crs \leftarrow \Pi_{\mathrm{NIZK}}.\mathrm{Setup}(1^\lambda)$ is the common reference string, s denotes statements belonging to an NP language $\mathcal{L}$, ω denotes witnesses s.t. $(s, \omega) \in \mathcal{R}_{\mathcal{L}}$. For a pair of polynomial time algorithm $(\mathcal{P}, \mathcal{V})$, $(\mathcal{P}, \mathcal{V})$ is called the non-interactive zero-knowledge argument of knowledge for language $\mathcal{L}$ if the following conditions are met.

Completeness: for any statement $s \in \mathcal{L}$ and a negligible function $\mathrm{negl}(\cdot)$, $\Pr[\mathcal{V}(s, \pi \leftarrow \mathcal{P}(s, \omega)) = 1] \geq 1 - \mathrm{negl}(\lambda)$.

Knowledge Soundness: for any statement $s' \notin \mathcal{L}$, any adversary $\mathcal{P}'$ and a knowledge extractor $\mathcal{Ext}(\cdot)$:

$$\Pr\left[\mathcal{V}\left(s', \pi' \leftarrow \mathcal{P}'(s', \omega')\right)\right] = 1 < \mathrm{negl}(\lambda) \wedge$$
$$Pr[\mathcal{Ext}(crs, \pi', s') \to \omega'] \geq 1 - \mathrm{negl}(\lambda).$$

Zero-knowledge: for any statement $s \in \mathcal{L}$, a zero-knowledge simulator $\mathcal{S}$ with simulation trapdoor τ , $\mathcal{P}(s, \omega) \overset{c}{\approx} \mathcal{S}_\tau(s) \wedge$ $\Pr[\mathcal{S}_\tau \text{ fails}] < \mathrm{negl}(\lambda)$.

2.4 Our Construction

Let $\mathcal{E}$ be an encryption scheme with the properties defined above, and Π_{NIZK} be a NIZK argument system. A feasible construction is as follows.

• $\mathrm{Setup}\left(1^\lambda\right)$: Taking as input a security parameter λ, and generates public parameters pp_{sv}.
• $\mathrm{KeyGen}(pp_{sv})$: Taking as input the public parameters pp_{sv}, each candidate $\mathbb{c}_j \in \mathbb{C}$ performs the key generation algorithm of $\mathcal{E}$, to generate the public encryption key pk_j and secret key sk_j.(If necessary, such as to defend against the rogue public-key attack[126], proving knowledge of the secret key must be performed in this algorithm.).
• $\mathrm{Vote}\left(pp_{sv}, \{pk_j\}_{j\in[\mathbb{C}]}, \vec{v}_i, w_i\right)$: Taking as input the public parameter pp_{sv}, the aggregated encryption key $ek \coloneqq \prod_j^{[\mathbb{C}]} pk_j$, a vector $\vec{v}_i$ of ballots and the weight w_i of voter $\mathbb{v}_i$, each voter $\mathbb{v}_i$ outputs ciphertext $\overrightarrow{ct}_i \coloneqq \left(ct_{i,j}^0 \coloneqq \mathcal{E}.\mathrm{Comm}(r_{i,j}), ct_{i,j}^1 \coloneqq \mathcal{E}.\mathrm{Enc}\left(ek, v_{i,j} \cdot w_i; r_{i,j}\right)\right)_{j\in[\mathbb{C}]}$, where $v_{i,j} \in \vec{v}_i$ denotes the value of the voter $\mathbb{v}_i$ casting for the candidate $\mathbb{c}_j$.

▪ $\mathrm{VoteAgg}\left(\mathbb{CT}', \overrightarrow{ct}_i\right)$: Upon receiving the ballot ciphertexts $\overrightarrow{ct}_i$ from a voter $\mathbb{v}_i$, the aggregator $\mathbb{a}$ updates its state variable set $\mathbb{CT} = \left\{\left(\mathbb{CT}.CT_j^0 = \mathbb{CT}'.CT_j^0 \cdot ct_{i,j}^0, \mathbb{CT}.CT_j^1 = \mathbb{CT}'.CT_j^1 \cdot ct_{i,j}^1\right)\right\}_{j\in[\mathbb{C}]}$. For simplicity, set $\mathbb{CT}.\mathbb{C}_0 \coloneqq \left\{CT_j^0\right\}_{j\in[\mathbb{C}]}$, $\mathbb{CT}.\mathbb{C}_1 \coloneqq \left\{CT_j^1\right\}_{j\in[\mathbb{C}]}$.

• $\mathrm{DKeyGen}(sk_j, \mathbb{CT}.\mathbb{C}_0)$: When the voting phase finished, each candidate $\mathbb{c}_j \in \mathbb{C}$ takes the aggregated set $\mathbb{CT}.\mathbb{C}_0$ and secret key sk_j as inputs, then generates the partial decryption keys $\overrightarrow{dk}_j \coloneqq \left\{dk_{j,j'} \coloneqq \mathcal{E}.\mathrm{KeyGen}\left(sk_j, CT_{j'}^0\right)\right\}_{j'\in[\mathbb{C}]}$ and an argument $\pi_j \leftarrow \Pi_{\mathrm{NIZK}}.\mathrm{Prove}(pp_{sv}, \overrightarrow{dk}_j, sk_j)$ to $\mathbb{a}$.

▪ $\mathrm{DKeyAgg}\left(\overrightarrow{dk}_j, \pi_j, \mathbb{DK}'\right)$: After receiving $\overrightarrow{dk}_j$ from each candidate $\mathbb{c}_j \in \mathbb{C}$, $\mathbb{a}$ checks whether $\Pi.\mathrm{Verify}\left(pp_{sv}, \overrightarrow{dk}_j, \pi_j\right)$ outputs 1. If positive, update the decryption key set $\mathbb{DK} = \left\{DK_{j'} = \mathbb{DK}'.DK_{j'} \cdot dk_{j,j'}\right\}_{j'\in[\mathbb{C}]}$; else return $\perp$.

• $\mathrm{Tally}(\mathbb{DK}, \mathbb{CT}.\mathbb{C}_1) \to \{t_j\}_{j\in[\mathbb{C}]}$: Takes as input the decryption key set $\mathbb{DK}$ and the aggregated ballot ciphertext set $\mathbb{CT}.\mathbb{C}_1$, outputs final tallies $\left\{t_j = \mathcal{E}.\mathrm{Dec}\left(DK_j, CT_j^1\right) =< \vec{v}_j, \vec{w} >\right\}_{j\in[\mathbb{C}]}$.

Correctness. By the LCH, LEH, and LKH properties, we prove the correctness of the construction in Appendix A.1.

Instantiation. The detailed instantiation building from the modified BC-IND secure encryption[110] scheme and Fiat-Shamir-based NIZK argument system [107,108] is shown in Appendix A.2.

2.5 Security Analysis

Ballot Privacy. The ballot privacy is guaranteed by the BC-IND-secure public key encryption. Informally speaking, the indistinguishability of ciphertexts means that ciphertexts cannot be distinguished from one another regardless of the ballot on which the ciphertext is generated. On the other hand, if an attacker can generate valid decryption keys without knowing candidates' secret keys, it can quickly reveal each voter's ballot, while the BC-IND-secure encryption scheme can prevent it.

Decryption-Key Verifiability. This property straightforwardly relies on the completeness and computational soundness of the NIZK argument system Π_{NIZK}, which means once the proof passes the Π_{NIZK}.Verify algorithm, the decryption key is correctly created by the owner of the secret key with overwhelming probability. On the contrary, if the decryption key is generated by a secret key independent of the candidate's public key, it passes the Π_{NIZK}.Verify algorithm with negligible probability.

To formally prove the properties, we first give the formal security definition of the construction for SemiVoting, then prove the corresponding theorem in Appendix A.3.

Definition 2. *Let $\mathcal{F}_{SV}$ be the functionality as defined below. We say that the construction in Section 2.4 securely computes $\mathcal{F}_{SV}$ in the presence of the static adversaries if for every PPT adversary $\mathcal{A}$ for the real world, there exists a PPT simulator $\mathcal{S}$ for the ideal world, such that:*

$$\mathrm{IDEAL}_{\mathcal{S}(z)}^{\mathcal{F}_{sv}}\left(\{\mathbb{v}_i: \vec{v}_i, w_i\}_{i\in[\mathbb{V}_c]}, \{\mathbb{c}_j: sk_j\}_{j\in[\mathbb{C}_c]}, \mathbb{a}: \perp\right) \overset{c}{\approx} \mathrm{REAL}_{\mathcal{S}(z)}^{\Pi_{SV}}\left(\{\mathbb{v}_i: \vec{v}_i, w_i\}_{i\in[\mathbb{V}_c]}, \{\mathbb{c}_j: sk_j\}_{j\in[\mathbb{C}_c]}, \mathbb{a}: \perp\right)$$

In Definition 2, we consider a static adversary $\mathcal{A}$ who controls one of the two parties: semi-honest voters and malicious candidates. We denote corrupted sets of voters and candidates by $\mathbb{V}_c$ and $\mathbb{C}_c$, respectively, and $|\mathbb{C}_c| \le \delta_c, |\mathbb{V}_c| \le \delta_v$. Intuitively, our construction is secure against them if the semi-honest voters and malicious candidates only learn the encrypted ballots $\{\vec{ct}_i\}_{i\in[\mathbb{V}\setminus\mathbb{V}_c]}$ and tally result $\{t_j\}_{j\in[\mathbb{C}]}$ nothing about unencrypted ballots $\{\vec{v}_i, w_i\}_{i\in[\mathbb{V}\setminus\mathbb{V}_c]}$. Besides, the internal state variables of aggregator $\mathbb{a}$ is append-only, transparent and verifiable for any participant. We formalize the definition based on the simulation paradigm. For convenience, let: $\mathcal{F}_{\mathrm{SV}}: \{\vec{v}_i, w_i\}_{i\in[\mathbb{V}]} \times \{sk_j\}_{j\in[\mathbb{V}]} \times \perp \rightarrow \{\vec{ct}_i\}_{i\in[\mathbb{V}]} \times \{\vec{dk}_j, \pi_j\}_{j\in[\mathbb{C}]} \times \{t_j\}_{j\in[\mathbb{C}]}$ be the functionality for the SemiVoting construction Π_{SV}, where $\perp$ refers to an empty string. For each tuple of inputs $\{\vec{v}_i, w_i\}_{i\in[\mathbb{V}]}$ belonging to voters and $\{sk_j\}_{j\in[\mathbb{V}]}$ belonging to candidates, the function $\mathcal{F}_{\mathrm{SV}}$ outputs $\{\vec{ct}_i\}_{i\in[\mathbb{V}]}$, $\{\vec{dk}_j, \pi_j\}_{j\in[\mathbb{C}]}$ and $\{t_j\}_{j\in[\mathbb{C}]}$ to voters and candidates, respectively. To define security, we first formalize the ideal and real worlds. In the ideal world, a non-corrupted trusted third party (TTP) also computes the functionality on the inputs and forwards to each party its respective output. It is said to achieve privacy and security if there exists a simulator $\mathcal{S}$ who can emulate the execution of the real world in the ideal world and no one can distinguish them.

Real world. Our construction Π_{SV} is conducted among the parties in $\mathbb{V}, \mathbb{C}$ and the static adversary $\mathcal{A}$. At the beginning of the construction Π_{SV}, each voter $\mathbb{v}_i \in \mathbb{V}$ first receive her input $(\vec{v}_i, w_i)$ then receive random coins r_i and an auxiliary input z. At the end of the execution, the honest party outputs whatever is prescribed by the protocol Π_{SV} and the adversary $\mathcal{A}$ outputs its view. The output of the real-world execution of the protocol Π_{SV} among these parties in the presence of the adversary $\mathcal{A}$ is defined as $\mathrm{REAL}_{\mathcal{A}(z)}^{\Pi_{\mathrm{SV}}}(\{\mathbb{v}_i: \vec{v}_i, w_i\}_{i\in[\mathbb{V}_c]}, \{\mathbb{c}_j: sk_j\}_{j\in[\mathbb{C}_c]}, \mathbb{a}: \perp)$.

Ideal world. The ideal world is executed among the parties $\mathbb{V}, \mathbb{C}$ and a simulator $\mathcal{S}$ who is allowed to corrupt the aggregator $\mathbb{a}$ and some subset of voters or candidates. At the beginning of the ideal world, each party receives the same input as the corresponding party in the real world. The honest party always forwards its input to the TTP. The corrupted party may abort or send arbitrary input. The TTP returns the result $\vec{ct}_i$ to the voter $\mathbb{v}_i$ or $(\vec{dk}_j, \pi_j)$ to the candidate $\mathbb{c}_j$. If the TTP receives an abort message as input, it sends the abort message to the voter. The output of the parties in the ideal world in the presence of the simulator $\mathcal{S}$ is defined as $\mathrm{IDEAL}_{\mathcal{S}(z)}^{\mathcal{F}}(\{\mathbb{v}_i: \vec{v}_i, w_i\}_{i\in[\mathbb{V}_c]}, \{\mathbb{c}_j: sk_j\}_{j\in[\mathbb{C}_c]}, \mathbb{a}: \perp)$.

Then, we have the below theorem.

Theorem 1. *Suppose the encryption scheme is BC-IND-secure, the NIZK argument system is computationally sound, and the internal state variables of aggregator $\mathbb{a}$ are append-only, transparent, and verifiable for any participant, then our SemiVoting construction is (δ_c, δ_v)-bounded secure as defined in Definition 2.*

Rather than the above properties, our construction also captures the features below.

Decentralization. Based on **Fact 1** and **Fact 2** in the introduction, we distribute the power of generating partial decryption keys to all candidates, meaning no single entity or organization can influence the outcome of elections. Furthermore, we employ blockchain technology to enhance decentralization in Section 3.

Self-Tallying. This functionality implies *public verifiability*, as anyone, including external and internal observers, can calculate the election outcome by themselves, i.e., by taking the aggregated decryption keys set $\mathbb{DK}$ and ciphertexts $\mathbb{CT}$ as inputs and running the Tally algorithm to get the output.

Moderately Succinctness. We exploit the nature of linear homomorphic encryption, which allows us to tally the ballot results without revealing the content of ballots. In addition, the cost of the Tally and Vote algorithms is only related to the number of candidates $|\mathbb{C}|$, not the number of voters $|\mathbb{V}|$.

Dynamic join. Since the initialization of public system parameters and the generation of decryption keys are independent of voters, any voter can dynamically join the voting system and generate encrypted ballots using system parameters before the voting phase finishes.

Although SemiVoting satisfies the described properties, it is still a strawman construction, as it lacks the restriction of dishonest voters' illegal behaviors and does not provide anonymity for voters. To overcome these issues, we propose VeriVoting.

Table 2: Symbols

Symbols	Meaning
vsk_i	A random seed key sampled by $\mathbb{v}_i$
$\mathrm{PRF}_{vsk}(\cdot)$	Pseudorandom Function
cm_i	Commitment on $\mathbb{v}_i$'s token(i.e. weight)
$addr_i$	Voter $\mathbb{v}_i$'s wallet address
$\mathbb{Com}$	Set of token commitments
$\mathbb{L}$	Append-only ledger storing published transactions
$\mathbb{MT}$	Merkle tree over token commitments. The depth of the Merkle tree denotes by $\mathbb{MT}$.depth
ρ_i	The authentication path from $(addr_i, cm_i)$ to Merkle tree's root $\mathbb{MT}$.rt.
$\mathbf{P_{vote}}(\cdot)$	Compiled program of the voting function
$\mathbf{F_{vote}}(\cdot)$	The prescribed voting function
crs_{PK}, crs_{VK}	Proving keys and Verification keys of zkSNARKs

3 VERIVOTING SCHEME

In this section, we remove the assumption that the voter is semi-honest in SemiVoting, such that voters can be dishonest participants who do not follow the prescribed voting rule. To prevent dishonest voters from repeating voting and casting the ill-formed ballot, while keeping voters' identities private, we employ zkSNARKs [32] and blockchain[44] to present our weighted voting scheme, VeriVoting, which implements and extends the SemiVoting primitive. For simplicity, we introduce some new symbols used in VeriVoting in Table 2.

3.1 Building Blocks

ZkSNARKs. Zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs)[32]-[34] are generic NIZK argument constructions supporting any arithmetic circuit φ and featuring constant-cost proof verification in the size of φ. Due to their low verification costs, zkSNARKs are frequently used on the Ethereum blockchain [35],[36]. Consistent with the traditional NIZK argument system[37]-[39], a zkSNARK system Π_{SNARK} also consists of two core algorithms: $\Pi_{\mathsf{SNARK}}.\mathsf{Prove}(crs_{PK}, s, \omega) \to \pi$ and $\Pi_{\mathsf{SNARK}}.\mathsf{Verify}(crs_{VK}, s, \pi) \to 1/0$, and meet completeness, knowledge soundness and zero-knowledge properties.

Another important reason we use zkSNARKs is that it can make our scheme support more flexible voting functions or complex ballot condition formats as long as they can be compiled into an arithmetic circuit.

PRFs. A pseudorandom function family[40,41] $\mathsf{PRF}_{sk}\colon \{0,1\}^{|x|} \to \{0,1\}^{\lambda(|x|)}$, where sk denotes the seed key, $x \in \{0.1\}^*$, $\lambda\colon \mathbb{N} \to \mathbb{N}$, is computationally indistinguishable from a random function family.

Commitments. Commitment schemes[42,43] are designed so that a party cannot change the value or statement after they have committed to it: that is, commitment schemes are binding, while keeping the value or statement hidden to others: that is, commitment schemes are hiding. Let Comm denote a statistically-hiding non-interactive commitment scheme, i.e., $cm \coloneqq \mathsf{Comm}(w, s)$, where w is the token, s is randomness; Then, cm is opened by revealing w and s, and one can verify that $\mathsf{Comm}(w, s)$ equals cm.

Smart Contracts. Smart contracts are a core building block of decentralized applications, facilitating the execution of specified tasks (such as payments) based on predetermined rules. Since the smart contract features transparency, verifiability, trust, and accuracy[44], we exploit it as the verifier to verify the output of parties in our VeriVoting scheme and as the aggregator to combine received ciphertexts and partial decryption keys.

3.2 Overview

The overview of the VeriVoting scheme is shown in Figure 2. In addition to the entities already defined in the SemiVoting primitive, we introduce two new entities: election authority and smart contract, where the smart contract plays the aggregator role.

Election Authority($\mathcal{EA}$). The responsibilities of $\mathcal{EA}$ include generating global public parameters, accepting voters' requests for registration, issuing tokens (equivalent to the weight value of different roles) to authenticated voters, and publishing the list of eligible voters and the list of candidates.

Smart Contract($\mathcal{SC}$). The functions of $\mathcal{SC}$ include uploading each voter's submitted commitment and secret ballot vector into the blockchain after verifying the validity of the commitments and ballots in the commitment and voting phase, uploading each candidate's submitted partial decryption keys into the blockchain after verifying the correctness of the keys in the DK generation phase.

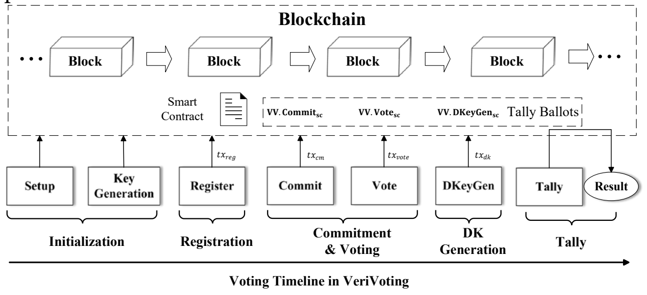

Figure 2: Overview of VeriVoting

Definition 3 (VeriVoting, VV). A VeriVoting scheme, over a set of voters $\mathbb{V}$, a set of candidates $\mathbb{C}$, smart contracts $\mathcal{SC}$s, and an election authority $\mathcal{EA}$, is defined by three algorithms and four protocols:

- $(pp_{sv}, crs) \leftarrow \mathsf{Setup}\left(1^\lambda\right)$ Input security parameter λ, call $\mathsf{SV.Setup}(1^\lambda)$ and $\Pi_{\mathsf{SNARK}}.\mathsf{Setup}(1^\lambda)$, outputs system public parameters pp_{sv} and $crs = (crs_{PK}, crs_{VK})$.
- $(pk_j, sk_j) \leftarrow \mathsf{KeyGen}_{\mathbb{c}}(pp_{sv})$ Each candidate $\mathbb{c}_j \in \mathbb{C}$ takes as input the public parameters pp_{sv}, calls $\mathsf{SV.KeyGen}(pp_{sv})$ to output (pk_j, sk_j).
- $vsk_i \leftarrow \mathsf{KeyGen}_{\mathbb{v}}(pp_{sv})$. Each voter $\mathbb{v}_i \in \mathbb{V}$ takes as input the public parameters pp_{sv}, outputs a random seed key vsk_i as the secret key of the PRF.
- $(cred_i, addr_i) \times tx_{reg} \leftarrow \mathsf{Register}(\mathbb{v}_i\colon vsk_i \times \mathcal{EA}\colon (cred_i, addr_i))$.
  - Each voter $\mathbb{v}_i \in \mathbb{V}$ submits her valid credential $cred_i$ and an empty wallet address $addr_i$ generated by vsk_i to $\mathcal{EA}$.
  - After verifying the validity of the voter's credential, $\mathcal{EA}$ transfers a token w_i (equals the voter's weight value) to the voter's address, which can be expressed as a transaction $tx_{reg} \coloneqq (addr_{\mathcal{RA}} \to addr_i\colon w_i)$.
- $tx_{cm} \times 1/0 \leftarrow \mathsf{Commit}(\mathbb{v}_i\colon (pp_{sv}, w_i, addr_i) \times \mathcal{SC}\colon (pp_{sv}, tx_{cm}, \mathbb{L}))$.
  - Each voter $\mathbb{v}_i \in \mathbb{V}$ takes the public parameters pp_{sv} and a token w_i as inputs, performs the Commit algorithm to generate a commitment on the token w_i, $cm_i \coloneqq \mathsf{Comm}(w_i, s_i)$, then outputs the transaction $tx_{cm} \coloneqq (addr_i, cm_i, w_i, s_i)$ to $\mathcal{SC}$.
  - Once called by the voter $\mathbb{v}_i$, the contract $\mathcal{SC}$ parses tx_{com} as $(addr_i, cm_i, w_i, s_i)$, computes $cm_i' \coloneqq \mathsf{Comm}(w_i, s_i)$ and executes the following assert:

    Assert $cm_i' = cm_i$;
    Assert $(addr_i, coin_i) \notin \mathbb{Comm}$;
    Assert $\mathbb{L}[addr_i] = w_i$.

    Then, $\mathcal{SC}$ sets $\mathbb{L}[addr_i] = 0$ and adds $(addr_i, cm_i)$ to $\mathbb{Comm}$.
- $tx_{vote} \times 1/0 \leftarrow \mathsf{Vote}(\mathbb{v}_i\colon (crs_{PK}, pp_{sv}, \{pk_j\}_{j\in[\mathbb{C}]}, vsk_i, w_i, \vec{v}_i, \mathbf{P_{vote}}, \mathbb{MT}.\mathsf{rt}, \mathbb{MT}.\mathsf{depth}, \rho_i) \times \mathcal{SC}\colon (crs_{VK}, tx_{vote}))$.
  - Each voter $\mathbb{v}_i \in \mathbb{V}$ firstly takes public parameters pp_{sv} and candidates' public keys $\{pk_j\}_{j\in[\mathbb{C}]}$, her seed key vsk_i and token w_i, and $(\mathbb{MT}.\mathsf{rt}, \mathbb{MT}.\mathsf{depth}, \rho_i)$ as inputs of the compiled

voting program $\mathbf{P_{vote}}$, runs $\mathbf{P_{vote}}$ to output statement $\mathbf{s}_i$ and witness $\mathbf{a}_i$. Then $\mathbb{v}_i$ performs $\Pi_{\text{SNARK}}.\mathsf{Prove}(crs_{PK}, \mathbf{s}_i, \mathbf{a}_i)$ to generate proof π_i, and finally sends $tx_{vote} \coloneqq (\mathbf{s}_i, \pi_i)$ to $\mathcal{SC}$.

The detail of the compiled program $\mathbf{P_{vote}}$, which contains $\mathsf{SV.Vote}$ algorithm, is shown in Figure 3.

▪ Once called by the voter $\mathbb{v}_i$, the contract $\mathcal{SC}$ parses tx_{vote} as $(\mathbf{s}_i, \pi_i)$, then runs $\Pi_{\text{SNARK}}.\mathsf{Verify}(crs_{VK}, \mathbf{s}_i, \pi_i)$ to check the validity of statements. If the verification passes, $\mathcal{SC}$ performs $\mathsf{SV.VoteAgg}$ algorithm to update $\mathbb{CT}$.

- $tx_{dk} \times 1/0 \leftarrow \mathsf{DKeyGen}\big(\mathbb{c}_j: (sk_j, \mathbb{CT}.\mathbb{C}_0) \times \mathcal{SC}: (pp_{sv}, tx_{dk})\big)$.

▪ Each candidate $\mathbb{c}_j \in \mathbb{C}$ executes $\mathsf{SV.DKeyGen}$ algorithm and sends $tx_{dk} \coloneqq (\overrightarrow{dk}_j, \pi_j)$ to $\mathcal{SC}$.

▪ The contract $\mathcal{SC}$ parses tx_{vote} as $\big(\overrightarrow{dk}_j, \pi_j\big)$, and performs $\mathsf{SV.DKeyAgg}$ algorithm to verify-then-update $\mathbb{DK}$.

- $\{t_j\}_{j\in[\mathbb{C}]} \leftarrow \mathsf{Tally}(\mathbb{DK}, \mathbb{CT}.\mathbb{C}_1)$. Any participant can carry out the $\mathsf{SV.Tally}$ algorithm to output final weighted tallies$\{t_j =< \vec{v}_j, \vec{w} >\}_{j\in[\mathbb{C}]}$.

```
Compiled Program P_vote(·)
INPUTS:
– public parameters $pp_{sv}$
– public keys $\{pk_j\}_{j\in[\mathbb{C}]}$
– Voter $\mathbb{v}_i'$s weight $w_i$ and $cm_i$
– Voter $\mathbb{v}_i'$s seed key $vsk_i$
– Ballot vector $\vec{v}_i$
– Path $\rho_i$ and $\mathbb{MT}.\mathrm{rt}$, $\mathbb{MT}.\mathrm{depth}$
OUTPUTS:
– Statement $\mathbf{s}_i \coloneqq (\overrightarrow{ct}_i, sn_i)$
– Witness $\mathbf{a}_i \coloneqq (\rho_i, vsk_i, \vec{v}_i, w_i, \star)$, where $\star$ are witnesses of computa-
    tion.
1. Assert $\mathsf{Predicate}(\vec{v}_i) = \mathsf{true}$.
2. Compute $sn_i \coloneqq \mathrm{PRF}_{vsk_i}(addr_i \coloneqq \mathrm{PRF}_{vsk_i}(0) \parallel cm_i)$.
3. Assert $\mathsf{MerkleAuth}(\mathbb{MT}.\mathrm{rt}, \mathbb{MT}.\mathrm{depth}, \rho_i, (addr_i, cm_i))$.
4. Compute ballot ciphertext vector:
        $\overrightarrow{ct}_i \coloneqq \mathsf{SV.Vote}(pp_{sv}, \big(\{pk_j\}_{j\in[\mathbb{C}]}, \vec{v}_i, w_i\big)$.
5. Output $\mathbf{s}_i$ and $\mathbf{a}_i$.
```

Figure 3: The compiled program $\mathbf{P_{vote}}(\cdot)$ (Removing the grey ground parts in Fig.3 is the definition of the function $\mathbf{F_{vote}}$.)

3.3 Security Requirements

To fix the vulnerabilities of SemiVoting, our VeriVoting scheme should follow the security requirements below.

Anonymity. In the voting phase, the voting transaction tx_{vote} cannot leak any information about a voter's identity, even if all the candidates collude with at most δ_v dishonest voters. For weighted voting, we assume that at least two honest voters with the same weight value cast their ballots for different candidates; otherwise, any observer can trace the voting transaction to the voter who owns the unique weight value.

Ballot Verifiability. To forbid dishonest voter's illegal behaviors, there are five statements that each voter should prove in the voting phase:

Statement 1: "$\mathsf{Predicate}(\vec{v}_i) = \mathsf{true}$", which means the ballot vector $\vec{v}_i$ should meet predefined voting predicate logic, e.g., $\mathsf{Predicate}(\vec{v}_i) \coloneqq (\vec{v}_i \in \{0,1\}^{|\mathbb{C}|}) \wedge (\parallel \vec{v}_i \parallel = 2)$ indicates that if each element in $\vec{v}_i$ should be in $\{0,1\}$ and the norm of $\vec{v}_i$ equals 1, output true; else false. As such, suppose a voter $\mathbb{v}_i$ is willing to cast her ballot for the candidate $\mathbb{c}_2$ in the three-candidate election, the well-formedness of the ballot vector should be $\vec{v}_i \coloneqq (0,1,1)$ that can pass the predicate function.

Statement 2: "$addr_i = \mathrm{PRF}_{vsk_i}(0)$", which means the voter $\mathbb{v}_i$'s seed key vsk_i matches the wallet address $addr_i$.

Statement 3: "$sn = \mathrm{PRF}_{vsk_i}(addr_i \parallel cm_i)$", which means that the serial number sn_i is computed correctly to prevent duplicate voting of dishonest voters.

Statement 4: "$\rho_i \in \mathbb{MT}$", which means the weight commitment $(addr_i, cm_i)$ appears as a leaf of a Merkle tree with root $\mathbb{MT}.\mathrm{rt}$.

Statement 5: "$\overrightarrow{ct}_i \leftarrow \mathsf{SV.Vote}(\cdot)$", the ballot ciphertexts are well-formed: for any ballot ciphertext $ct_{i,j} \in \overrightarrow{ct}_i$ it holds that $ct_{i,j} = \mathrm{Enc}(ek, v_{i,j} \cdot w_i; r_{i,j})$.

3.4 Our Construction

This section presents the construction of VeriVoting in six phases:

Initialization phase. In this phase, all public parameters are fixed and submitted onto the blockchain by the election authority $\mathcal{EA}$, candidates, and voters.

$\mathsf{VV.Setup}_{\mathrm{EA}}(1^\lambda, \mathbf{F_{vote}})$: $\mathcal{EA}$ performs $\mathsf{SV.Setup}(1^\lambda) \rightarrow pp_{sv}$ and $\Pi_{\text{SNARK}}.\mathsf{Setup}(1^\lambda, \mathbf{F_{vote}}) \rightarrow crs$, publishes public parameters pp_{sv} and common reference string $crs = (crs_{PK}, crs_{VK})$ on the blockchain, which serves as a public bulletin board.

$\mathsf{VV.KeyGen}_{\mathbb{c}}(pp_{sv})$: Each candidate $\mathbb{c}_j \in \mathbb{C}$ takes as input the public parameters pp_{sv}, calls $\mathsf{SV.KeyGen}(pp_{sv}) \rightarrow (pk_j, sk_j)$ and publishes pk_j on the blockchain.

$\mathsf{VV.KeyGen}_{\mathbb{v}}(pp_{sv})$: Each voter $\mathbb{v}_i \in \mathbb{V}$ takes as input the public parameters pp_{sv}, samples randomly a secret seed vsk_i, then outputs $addr_i \coloneqq \mathrm{PRF}_{vsk_i}(0)$ as her new wallet address.

For simplicity, we assume that the verification process of each candidate's pk_j was done before issuing them on the blockchain. In addition, if our NIZK system requires honestly generated CRSs, such as zkSNARKs, there are several well-known mechanisms that are practical with the parameters we need, such as distributed generation of crs by multiple parties[45] or by SGX[46].

Registration phase. In this phase, each voter registers his/her voting information with $\mathcal{EA}$ who will issue tokens(i.e., weight value) according to the role of each voter.

$\mathsf{VV.Register}_{\mathbb{v}}(cred_i, addr_i)$: Each voter $\mathbb{v}_i$ submits his/her valid voting credential $cred_i$ and empty wallet address $addr_i$ to $\mathcal{EA}$.

$\mathsf{VV.Register}_{\mathrm{EA}}(cred_i, addr_i)$: After verifying the validity of the voter's credential $cred_i$, $\mathcal{EA}$ transfers a token w_i equals the voter's weight to the voter's address $addr_i$, then adds the transaction $tx_{reg} \coloneqq (addr_{\mathcal{EA}} \Rightarrow addr_i: w_i)$ to the blockchain.

Commitment and voting phase. The formal descriptions of commitment and voting phases are given in Figure 4. We use the pseudocode format to illustrate the core phase of our construction.

In the commitment phase, each voter $\mathbb{v}_i \in \mathbb{V}$ needs to generate a commitment cm_i to his/her token w_i as the certificate of voting, deriving his/her weight from the token sent by $\mathcal{EA}$. Subsequently, all commitments of $\mathbb{Comm}$ on the ledger are collected in a Merkle tree, which facilitates efficiently proving that an address-commitment pair $(addr_i, cm_i)$ appears on the ledger.

In the voting phase, each voter $\mathbb{v}_i \in \mathbb{V}$ needs to convince $\mathcal{SC}$ that *Statements* 1-5 are correct. To do so, $\mathbb{v}_i$ performs $\mathbf{P_{vote}}(\cdot)$, which is the compiled version of the function $\mathbf{F}_{\mathbf{vote}}(\cdot)$ in SNARK-friendly format, and then calls $\Pi_{\text{SNARK}}.\text{Prove}$ algorithm to compute the proof π_i according to the outputs of $\mathbf{P}_{\mathbf{vote}}(\cdot)$ (s.t. $(\mathbf{s}_i, \mathbf{a}_i) \in \mathcal{R}_{\mathcal{L}_v}$, where $\mathcal{L}_v$ is the language consisting of valid voting statements, $\mathcal{R}_{\mathcal{L}_v}$ is the relation specifying a collection of valid statement-witness pairs $(\mathbf{s}_i, \mathbf{a}_i)$.

Possession of token being voted. It is essential for a voter to prove the possession of the on-chain token such that no one can illegally invade her token to vote. To prove this point efficiently, the blockchain maintains a Merkle tree $\mathbb{MT}$ over $\mathbb{Comm}$, which can be viewed as an oblivious Merkle tree construction to hide the voter of a token, and which token was transferred. Membership in the set can be demonstrated by a Merkle authentication path ρ_i consistent with the root hash $\mathbb{MT}.rt$; moreover, the serial number sn_i is computed using the same seed key vsk_i from which the address $addr_i$ is derived; these are done in zero-knowledge.

No duplicate voting. Each commitment $(addr_i, cm_i)$ has a cryptographically unique serial number sn_i that can be computed as a pseudorandom function of $\mathbb{v}_i$'s seed key vsk_i and commitment. To cast a ballot, its serial number sn_i must be disclosed, and a zero-knowledge proof must be given to show the correctness of sn_i. The contract checks that no sn_i is used twice.

Ballot ciphertexts aggregation. After verifying the correctness of tx_{vote} from a voter $\mathbb{v}_i$, the contract aggregate the new $\overrightarrow{ct_i}$ into $\mathbb{CT}$ by computing $\left\{CT_j^0 = CT_j^0 \cdot ct_{i,j}^0, CT_j^1 = CT_j^1 \cdot ct_{i,j}^1\right\}_{j\in[\mathbb{C}]}$. In other words, the state variables set $\mathbb{CT}$, stored in contract storage, is iteratively updated through the ciphertext aggregation when a new valid transaction arrives. At a high level, the whole process of aggregation can be expressed as:

$$\left\{(ct_{i,j}^0, ct_{i,j}^1)\right\}^{|\mathbb{V}|\times|\mathbb{C}|} \xrightarrow{\mathbf{agg}} \left\{(CT_j^0, CT_j^1)\right\}^{|\mathbb{C}|}.$$

Decryption-key(DK) generation phase. When the voting phase is over, $\mathcal{SC}$ publishes the aggregated ciphertexts set $\mathbb{CT}$ on the blockchain. By using their secret keys, each candidate creates partial decryption keys on $\mathbb{CT}.\mathbb{C}_0 \coloneqq \left\{CT_j^0\right\}_{j\in[\mathbb{C}]}$ and forwards the transaction $tx_{dk} \coloneqq (\overrightarrow{dk_j}, \pi_j)$ to $\mathcal{SC}$ who will check the validity of $\overrightarrow{dk_j}$ by corresponding proof π_j. After that, $\mathcal{SC}$ aggregates the received $\left\{\overrightarrow{dk_j}\right\}_{j\in[\mathbb{C}]}$, i.e., $\left\{dk_{j,j'}\right\}^{|\mathbb{C}|\times|\mathbb{C}|} \xrightarrow{\mathbf{agg}} \left\{dk_j\right\}^{|\mathbb{C}|}$, and then publishes $\mathbb{DK} \coloneqq \left\{dk_j\right\}^{|\mathbb{C}|}$ as the decryption key.

The above process of aggregating refers to SV.DKeyAgg algorithm where the aggregator $\mathbb{a}$ is replaced by $\mathcal{SC}$.

Smart Contract

```
**VV.Commit_sc**
INPUTS:
- Public parameters $pp_{sv}$
- A transaction $tx_{cm}$
- The ledger $\mathbb{L}$
OUPUTS bit $b$ (1-accept, 0-reject):
1. Prase $tx_{cm}$ as $(addr_i, cm_i, w_i, s_i)$.
2. Compute $cm_i' \coloneqq \text{COMM}(s_i, w_i)$.
3. Assert $cm_i' = cm_i$.
4. Assert $(addr_i, cm_i) \notin \mathbb{Comm}$.
5. Assert $\mathbb{L}[addr_i] = w_i$.
6. Set $\mathbb{L}[addr_i] = 0$.
7. Append $(addr_i, cm_i)$ to $\mathbb{Comm}$.
8. Return $b$. //if all success, $b = 1$ else 0.

**VV.Vote_sc**
INPUTS:
- Verification key $crs_{VK}$
- A transaction $tx_{vote}$
- The ledger $\mathbb{L}$
OUTPUTS: bit $b$ (1-accept, 0-reject)
1. Prase $tx_{vote}$ as $(\pi_i, \mathbf{s}_i)$, where $\mathbf{s}_i \coloneqq (\overrightarrow{ct}_i, sn_i)$.
2. Assert $\Pi_{\text{SNARK}}.\text{Verify}(crs_{VK}, \mathbf{s}_i, \pi_i)$.
3. Assert $sn_i \notin \mathbb{SN}$.
4. Add $sn_i$ to $\mathbb{SN}$.
5. Update $\mathbb{CT}$ by aggregating the received $\overrightarrow{ct}_i$.
6. Return $b$. //if all success, $b = 1$ else 0.
```

Voter

```
**VV.Commit_v**
INPUTS:
- Public parameters $pp_{sv}$
- The voter $\mathbb{v}_i$'s weight $w_i$
- The voter $\mathbb{v}_i$'s wallet address $addr_i$
OUTPUTS: transaction $tx_{cm}$
1. Randomly sample a randomness $s_i$.
2. Compute $cm_i \coloneqq \text{COMM}(s_i, w_i)$.
3. Store $(cm_i, s_i, w_i)$ in the wallet.
4. Send $tx_{cm} \coloneqq (addr_i, cm_i, w_i, s_i)$ to $\mathcal{SC}$.

**VV.Vote_v**
INPUTS:
- Public parameters $pp_{sv}$
- Proving key $crs_{PK}$
- Set of Candidates' public keys $\{pk_j\}_{j\in[\mathbb{C}]}$
- Voter $\mathbb{v}_i$'s weight $w_i$ and $cm_i$
- Voter $\mathbb{v}_i$'s seed key $vsk_i$
- Ballot vector $\vec{v}_i$
- Authentication path $\rho_i$ and $\mathbb{MT}.\text{rt}$, $\mathbb{MT}.\text{depth}$
- Compiled program $\mathbf{P}_{\mathbf{vote}}(\cdot)$
OUTPUTS: transaction $tx_{vote}$
1. Perform $\mathbf{P}_{\mathbf{vote}}(\cdot)$ to obtain statement $\mathbf{s}_i$ and witness $\mathbf{a}_i$.
2. Remove $(cm_i, s_i, w_i)$ from the wallet.
3. Compute $\pi_i \coloneqq \Pi_{\text{SNARK}}.\text{Prove}(crs_{PK}, \mathbf{s}_i, \mathbf{a}_i)$
4. Send $tx_{vote} \coloneqq (\pi_i, \mathbf{s}_i)$ to $\mathcal{SC}$.
```

Figure 4: The commitment and voting phase

Tally phase. Any participant can run the SV.Tally algorithm to compute the tally results $\left\{t_j = < \vec{v}_j, \vec{w} >\right\}_{j\in[\mathbb{C}]}$. For fairness, $\mathcal{EA}$ can call the contract $\mathcal{SC}$ to execute the SV.Tally$(\cdot)$ algorithm and publish the results on the blockchain. Notes that if we employ the instantiation implemented in Appendix A.3 as the encryption scheme, then the contract $\mathcal{SC}$ can call the off-chain computing power by Oracle[47] to compute the discrete logarithms $\left\{t_j'\right\}_{j\in[\mathbb{C}]}$ of $\left\{g^{t_j=<\vec{v}_j,\vec{w}>}\right\}_{j\in[\mathbb{C}]}$ and check if $\left(g^{t_j'} = g^{t_j=<\vec{v}_j,\vec{w}>}\right)_{j\in[\mathbb{C}]}$ holds.

Correctness. The correctness of the VeriVoting straightforwardly relies on the correctness of SemiVoting, the completeness of zkSNARKs, and the binding of the commitment scheme.

3.5 Security Analysis

Anonymity. Unlike anonymous payment systems[19,20], our scheme is not a cash transfer system, which means no malicious recipient can provide additional information, e.g., token value, to an adversary $\mathcal{A}$. With the benefit of the zero-knowledge feature of zkSNARKs, in the voting phase, an adversary $\mathcal{A}$ learns only the output $tx_{vote} \coloneqq (\pi_i, \mathbf{s}_i \coloneqq (\overrightarrow{ct_i}, \mathbb{MT}.rt, sn_i))$ which does not leak any information about the voter's identity, such that in the worst case, all candidates collude to decrypt the ballot ciphertext $\overrightarrow{ct_i}$ in order to recover $\vec{v}_i$, they cannot identify the voter who cast the revealed ballot $\vec{v}_i$.

Ballot Verifiability. With the soundness of zkSNARKs, no polynomial time adversary can prove that a false statement $s' \notin \mathcal{L}_v$, $\exists\, witness\ a'$, s.t. $(s', a') \in \mathcal{R}_{\mathcal{L}_v}$. Thus, ballot verifiability is ensured once the contract has verified the correctness of the transaction tx_{vote}.

To formally represent the security of VeriVoting, we first give the formal security definition of VeriVoting, then prove the corresponding theorem in Appendix B.1.

Definition 4. *Let $\mathcal{F}_{\text{VV}}$ be the functionality as defined below. We say that the construction in Section 3.4 securely computes $\mathcal{F}_{\text{VV}}$ in the presence of the static adversaries if for every PPT adversary $\mathcal{A}$ for the real world, there exists a PPT simulator $\mathcal{S}$ for the ideal world, such that*:

$$\text{IDEAL}^{\mathcal{F}_{\text{VV}}}_{\mathcal{S}(z)}\left(\{\mathbb{v}_i: \vec{v}_i, w_i, vsk_i\}_{i\in[\mathbb{V}_c]}, \{\mathbb{c}_j: \bot\}_{j\in[\mathbb{C}]}, \mathcal{SC}: \bot)\right) \overset{c}{\approx} \text{REAL}^{\Pi_{\text{VV}}}_{\mathcal{A}(z)}(\{\mathbb{v}_i: \vec{v}_i, w_i, vsk_i\}_{i\in[\mathbb{V}_c]}, \{\mathbb{c}_j: \bot\}_{j\in[\mathbb{C}]}, \mathcal{SC}: \bot)$$

Since VeriVoting is the extension of SemiVoting construction, the functionality $\mathcal{F}_{\text{VV}}$ is similar to $\mathcal{F}_{\text{SV}}$, except that the input/output of voters: $\mathcal{F}_{\text{VV}}: \{\vec{v}_i, w_i, vsk_i\}_{i\in[\mathbb{V}]} \times \bot \times \bot \rightarrow \{tx_{cm}, tx_{vote}\}_{i\in[\mathbb{V}]} \times \{tx_{dk}\}_{j\in[\mathbb{C}]} \times \{t_j\}_{j\in[\mathbb{C}]}$. Similar to Definition 2, to define the security, we first formalize the ideal and real worlds below.

Real world. Our construction Π_{VV} is conducted among the $\mathcal{SC}$ and parties in $\mathbb{V}, \mathbb{C}$ and the static adversary $\mathcal{A}$ who controls a subset of parties. At the beginning of the construction Π_{VV}, each voter $\mathbb{v}_i \in \mathbb{V}$ first receive his/her input $(\vec{v}_i, w_i, vsk_i)$ then receive random coins r_i and an auxiliary input z. At the end of the execution, the honest party outputs whatever is prescribed by the scheme Π_{SV} and the adversary $\mathcal{A}$ outputs its view. The output of the real-world execution of the protocol Π_{SV} among these parties in the presence of the adversary $\mathcal{A}$ is defined as

$$\text{REAL}^{\Pi_{\text{VV}}}_{\mathcal{A}(z)}(\{\mathbb{v}_i: \vec{v}_i, w_i, vsk_i\}_{i\in[\mathbb{V}_c]}, \{\mathbb{c}_j: \bot\}_{j\in[\mathbb{C}]} \mathcal{SC}: \bot).$$

Ideal world. The ideal world is executed among the $\mathcal{SC}$ and parties in $\mathbb{V}, \mathbb{C}$ and a simulator $\mathcal{S}$ who is allowed to corrupt the contract $\mathcal{SC}$ and some subset of voters or candidates. At the beginning of the ideal world, each party receives the same input as the corresponding party in the real world. The honest party always forwards its input to the TTP. The corrupted party may abort or send arbitrary input. The TTP returns the result tx_{cm} or tx_{vote} to the voter $\mathbb{v}_i \in \mathbb{V}$ or tx_{dk} to the candidate $\mathbb{c}_i \in \mathbb{C}$. If the TTP receives an abort message as input, it sends the abort message to the voter/candidate. The output of the parties in the ideal world in the presence of the simulator $\mathcal{S}$ is defined as

$$\text{IDEAL}^{\mathcal{F}_{\text{VV}}}_{\mathcal{S}(z)}(\{\mathbb{v}_i: \vec{v}_i, w_i, vsk_i\}_{i\in[\mathbb{V}_c]}, \{\mathbb{c}_j: \bot\}_{j\in[\mathbb{C}]}, \mathcal{SC}: \bot).$$

Then, we have the below theorem.

Theorem 2. *Suppose that the hash function in the Merkle tree is collision-resistant, the commitment scheme is perfectly binding and computationally hiding, the zkSNARK scheme is computationally zero-knowledge and simulation sound extractable, the PRF scheme is secure, and SemiVoting is (δ_c, δ_v)-bounded secure, then our scheme is secure as defined in Definition 4.*

4 PERFORMANCE ANALYSIS ON VERIVOTING

In this section, we first analyze the computation complexity of algorithms in four phases, including the commitment, voting, DK generation, and tally phases. Then we separately evaluate the computation cost of these algorithms on Remix IDE[25] and local computer with ZoKrates toolbox[26] by instantiating VeriVoting.

4.1 Analysis of Computation Complexity

Table 3 summarizes the computational complexity of algorithms running in different phases from a theoretical perspective and mainly focuses on the influence of the number of voters and candidates and the depth of the Merkle tree on algorithm complexity.

Table 3: Computational complexity of algorithms in VeriVoting

Phase	Entity	Algorithm	Complexity
Comm.	Voter	$\mathbf{Commit}_{\mathbb{v}}$	$O(1)$
	Contract	$\mathbf{Commit}_{\mathbf{sc}}$	$O(1)$
Voting	Voter	$\mathbf{Vote}_{\mathbb{v}}$	$O(\lvert\mathbb{C}\rvert + \mathbb{MT}.\text{depth})$
	Contract	$\mathbf{Vote}_{\mathbf{sc}}$	$O(\lvert\mathbb{C}\rvert + \mathbb{MT}.\text{depth})$
DK Gen.	Candidate	**DKeyGen**	$O(\lvert\mathbb{C}\rvert)$
	Contract	**DKeyAgg**	$O(\lvert\mathbb{C}\rvert)$
Tally	Contract	**Tally**	$O(\lvert\mathbb{C}\rvert)$

In the commitment phase, the voter straightforwardly commits her token regardless of the number of voters and candidates, i.e., the complexity of $\mathbf{Commit}_{\mathbb{v}}$ is trivially $O(1)$. Accordingly, the complexity of the verification algorithm $\mathbf{Commit}_{\mathbf{sc}}$ on the contract side is $O(1)$ as well.

In the voting phase, to guarantee ballot verifiability, the voter needs to make proof on *Statements* 1~5 in zero-knowledge. Obviously, in *Statements* 4 and 5, the depth of the Merkle tree and the number of candidates conduct the size of inputs of the function $\mathbf{F_{vote}}$. This implies that not only the complexity of the voting algorithm $\mathbf{Vote}_{\mathbb{v}}$, but also the size of crs depends on the depth of the Merkle tree $\mathbb{MT}.\text{depth}$ and the number of candidates $|\mathbb{C}|$. As a result, the complexity of the verification algorithm $\mathbf{Vote}_{\mathbf{sc}}$ taking crs_{VK} as input is $O(|\mathbb{C}| + \mathbb{MT}.\text{depth})$ as well.

In the DK generation phase, taking the aggregated ciphertexts set $\mathbb{CT}.\mathbb{C}_0$ as input, the candidate $\mathbb{c}_j$ generates partial decryption keys for each ciphertext in $\mathbb{CT}.\mathbb{C}_0$ and performs $\Pi_{\text{NIZK}}.\text{Proof}$ algorithm in **DKeyGen**. As the size of $\mathbb{CT}.\mathbb{C}_0$ is $|\mathbb{C}|$, the complexity of **DKeyGen** algorithm is $O(|\mathbb{C}|)$. Accordingly, taking the partial decryption key $(\overrightarrow{dk_j}, \pi_j)$ from candidate $\mathbb{c}_j$ as input, the contract needs to verify-then-aggregate $\overrightarrow{dk_j}$ $\left(|\overrightarrow{dk_j}| = |\mathbb{C}|\right)$ into $\mathbb{DK}$, the complexity of **DKeyAgg** algorithm is $O(|\mathbb{C}|)$ as well.

In the tally phase, since the tally process is indeed the decryption operation on $\mathbb{CT}$, the complexity of **Tally** algorithm is $O(|\mathbb{C}|)$ as the size of $\mathbb{CT}$ is $|\mathbb{C}|$.

Based on the above analysis, we conclude that the computational complexity of our scheme depends on the number of candidates and the depth of the Merkle tree, rather than the number of voters. Thus, we call our scheme *moderately succinct*, as the number of voters is always greater than that of candidates in most weighted voting schemes.

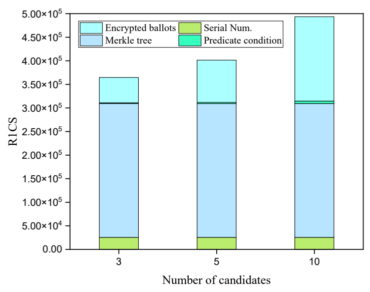

Figure 5: The number of constraints for $\mathbf{F}_{\mathbf{vote}}(\cdot)$

Table 4: Measurements for the commitment and voting phases

$\lvert\mathbb{C}\rvert$	Voter				Smart Contract		
	Commit$_{\mathbb{V}}$	**Vote**$_{\mathbb{V}}$			**Commit**$_{\mathbf{sc}}$	**Vote**$_{\mathbf{sc}}$	
	Time(ms)	Constraints (Num.)	crs_{PK} (MB)	Time(s)	Gas	crs_{VK} (KB)	Gas
3	0.686	364644	144.3	11.69	57160	27.2	1,254K
5		401476	160.9	15.05		44.2	1,362K
10		493556	197.9	18.96		86.7	1,630K

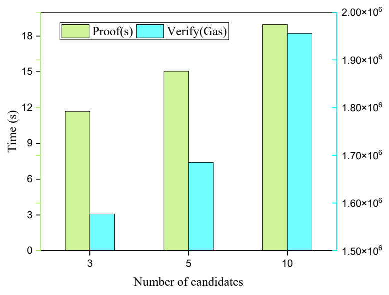

Figure 6: Comp./Gas cost of $\mathbf{Vote}_{\mathbb{V}}$ and $\mathbf{Vote}_{\mathbf{sc}}$

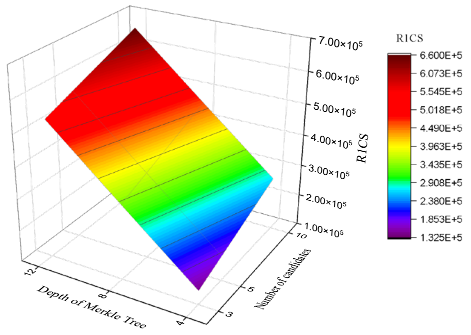

Figure 7: The number of constraints by two factors

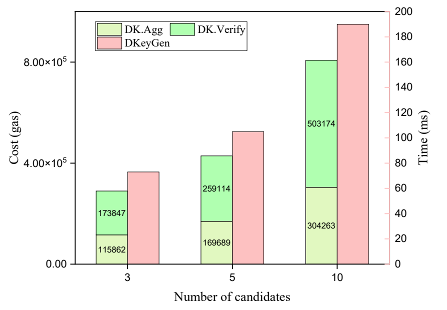

Figure 8: Comp./Gas cost of **DKeyGen** and **DKeyAgg**

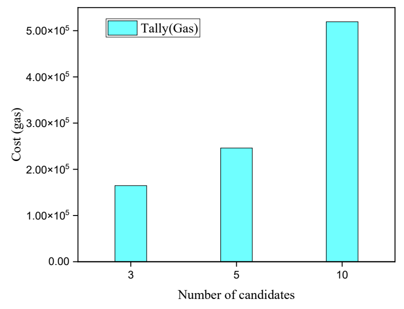

Figure 9: Gas cost of **Tally**

4.2 Performance Evaluation

In this section, we implement an instance of VeriVoting to evaluate the performance. The instantiation for SemiVoting has already been shown in Appendix A.3. The Commit, Merkle tree and PRF algorithms are implemented by SHA-256[48]. The zkSNARK system we chose is Groth16[32] with 128-bit security.

With the help of Remix IDE as the test platform for the contract and precompiled contracts for elliptic-curve operations[49] in Ethereum, we make some emulation tests[1] for our instantiation. The terminal configuration at the voter and candidate side is: Intel i7 7700HQ CPU, 8GB memory, Ubuntu OS with Zokrates toolbox.

We first present measurements for the commitment and voting phases in Table 4, in which we use 'Gas' to denote the contract cost (The gas cost includes the basic cost for sending a transaction, the storage cost, and the computational cost.). Due to the complexity of the voting phase is $O(\lvert\mathbb{C}\rvert + \mathbb{MT}.\text{depth})$, we let the depth of the Merkle tree be a bounded value, i.e., $\mathbb{MT}.\text{depth} = 8$, and apply different numbers of candidates, i.e., $\lvert\mathbb{C}\rvert \in \{3,5,10\}$ to evaluate the performance of each algorithm. In Table 4, The number of R1CS constraints is determined by the prescribed voting function $\mathbf{F}_{\mathbf{vote}}(\cdot)$ where the proportion is accounted for by the encryption operation $\mathcal{E}.\text{Encrypt}$ rises as the number of candidates increases, as shown in Figure 5. For the same reason, the size of crs, as well as time and gas costs of the voter and contract, grow linearly with the number of candidates, as shown in Figure 6. Furthermore, we take the Merkle tree's depth and the number of candidates as factors for emulating the number-changing trend of R1CS constraints. The emulation results are shown in Figure 7, where we set the depth of the Merkle tree varying from 4 to 12.

Table 5: Measurements for the DK generation phase

$\lvert\mathbb{C}\rvert$	Candidate		Smart Contract	
	DKeyGen		**DKeyAgg**	
	Π.Prove Time(ms)	DK Gen Time(ms)	Π.Verify (Gas)	DK Agg (Gas)
3	53	20	173847	115862
5	75	33	259114	169689
10	123	67	503174	304263

Next, we present measurements[2] for the decryption-key generation phase in Table 5, where we make independent statistics of the measurement results of the two algorithms, Π.Prove and Π.Verify, in the NIZK argument system Π to make explicit the computational cost accounted for by the argument system. The complete measurement results are drawn in Figure 8.

Finally, we present measurements for the tally phase in Figure 9. In this phase, we only evaluate the gas consumption of $\mathcal{E}$.Decrypt operation on different numbers of candidates, and do not count the statistics of computing the discrete logarithms.

5 A Distributed Voting Framework for the Large-Scale Election

A potential performance bottleneck we do not discuss in Section 4 is that the execution of smart contract transactions cannot be

[1] More details on Github: https://github.com/PropersonCyber/VeriVoting

[2] We use the BabyJubJub[127] curve to impletement DK generatioin and verification.

parallelized because Ethereum virtual machine (EVM) is a single-threaded state machine[50]. As a result, a single contract could not afford the requirement to execute a mass of transactions in a large-scale election. Although smart contracts functions can be migrated to the chaincode in Hyperledger Fabric[51] to considerably improve the performance[1,52,53] of executing transactions in VeriVoting, it is possible that a single party or organization may have the ability to control the consortium or private blockchain.

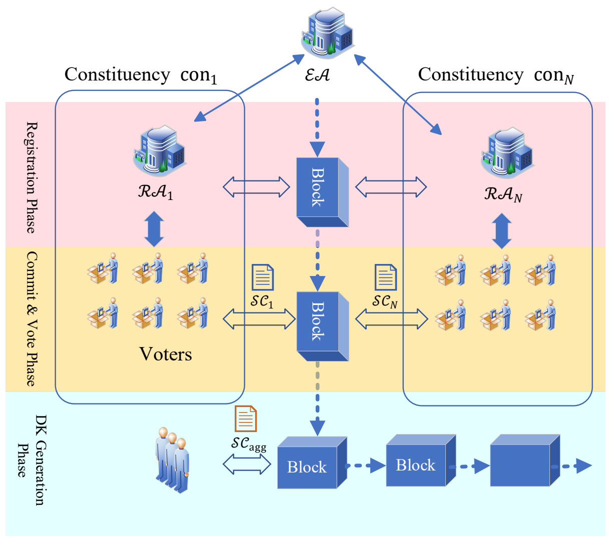

Figure 10: Overview of the distributed voting framework

By LCH, LEH, and LKH properties of our construction for SemiVoting, we can take a distributed approach to solve the challenge of being unable to execute in parallel. Figure 10 illustrates the overview of the distributed VeriVoting framework. We assume that the initialization phase is already done, i.e., all public parameters, a list of candidates, and smart contracts are published and deployed on the blockchain. In Figure 10, we divide the whole constituency into constituencies by some strategy(e.g., based on the voter's wallet address) in the registration phase, i.e., $\mathbf{con} \xrightarrow{\mathrm{div}} \{\mathsf{con}_k\}_{k\in[N]}$, where N is the number of constituencies. In addition, we introduce a new entity below:

• *Registration authority*($\mathcal{RA}$), located in each constituency, is responsible for accepting registration requests from voters instead of $\mathcal{EA}$ in the registration phase. The topology of $\mathcal{EA}$ and $\mathcal{RA}$ is similar to that of CA and RA in PKI. The difference is that CA issues digital certificates to users, whereas $\mathcal{EA}$ issues token to valid voters.

Next, we split the original functionality of contract $\mathcal{SC}$ into the following two parts:

• Smart contract $\mathcal{SC}_k$. In the commitment and voting phase, It is responsible for interacting with voters to verify the validity of commitments and ballots submitted by voters located in the constituency con_k. By the factory pattern in Solidity, $\mathcal{EA}$ can create an instance of the same contract for each constituency.

• Smart contract $\mathcal{SC}_{\mathrm{agg}}$. It is responsible for aggregating all aggregated ciphertext set $\{\mathbb{CT}_k\}_{k\in[N]}$ from constituencies and interacting with each candidate to generate decryption keys in the DK generation phase.

In Figure 10, the registration, commitment, and voting phases run as usual in each constituency. In the DK generation phase, $\mathcal{SC}_{\mathrm{agg}}$ calls the external contracts $\{\mathcal{SC}_k\}_{k\in N}$ to obtain each constituency's aggregated secret ballot set $\mathbb{CT}_k$, then aggregates all the set $\{\mathbb{CT}_k\}_{k\in N}$ into one set $\mathbb{CT} = (\mathbb{C}_0 = \prod_{k=1}^{N}\mathbb{CT}_k.\mathbb{C}_0, \mathbb{C}_1 = \prod_{k=1}^{N}\mathbb{CT}_k.\mathbb{C}_1) \in \mathcal{C}^{|\mathbb{C}|}$. After that, all candidates execute SV.DKeyGen algorithm with $\mathcal{SC}_{\mathrm{agg}}$ to output the decryption key set $\mathbb{DK}$. Finally, taking $\mathbb{CT}$ and $\mathbb{DK}$ as input, the SV.Tally algorithm outputs election results.

6 Discussion

At the beginning of the paper's introduction, we stressed that an e-voting scheme should meet decentralization, verifiability, and privacy-preserving properties, each of which is conducted by several practical performance or security and privacy requirements. In this section, we recall and show how VeriVoting achieves these required properties.

Decentralization. In decentralized e-voting systems, the two most essential requirements are performance and verifiability of the output of each entity in the scheme. The former requirement determines whether the voting scheme is suitable for large-scale elections and whether it supports voters dynamically joining the vote; the latter requirement determines the credibility of the election results.

In response to the performance requirement, the computational complexity of the main algorithms of the scheme, i.e., Commit and Vote algorithms, is related to the number of candidates. Although the number of executions is related to the number of voters for the contract in the commitment and voting phases, the performance can be optimized by a distributed method due to the ciphertext aggregatable feature of SemiVoting, as in Section 5. On the other hand, VeriVoting supports *dynamic join*, as none of the public parameters are generated by voters, such that voters can still join the election by the registration algorithm before the deadline of the voting phase. However, most other voting schemes [115]-[118] do not support the dynamic joining of voters, due to the system parameters need to be generated jointly by a fixed number of voters. In response to the need for verifiability, we next discuss verifiability.

Verifiability. With the *decryption-key and ballot verifiability*, the output of each voter and candidate can be publicly verified in the contract. With the append-only and transparency of the contract (i.e., the state variables and entire process of actions taken in a smart contract are publicly visible), voters are able to trivially verify that their ballots were actually counted by checking the change before and after the update of state variables. Moreover, with the *self-tallying* property[54], every observer, including voters, candidates, and election officials, is able to verify whether the final election outcome indeed corresponds to the votes submitted by the voters.

Privacy-Preserving. In the VeriVoting scheme, we decouple privacy-preserving on voter identity and ballot content to achieve *bidirectional unlinkability*. It is a stronger privacy-preserving notion that if either *ballot privacy* is broken or *anonymity* is broken, not both, the adversary cannot determine *by whom* the revealed ballot is cast or *which* candidates the revealed voter cast her ballot for. Our approach to decoupling is that ballot privacy is guaranteed by encryption schemes where the encryption key is generated by all candidates, and anonymity is ensured by zkSNARKs where the crs is created by different entities, such as SGX[46].

Rather than the above necessary properties, VeriVoting captures *eligibility*, *transparency*, *fairness*, *recipient-freeness*[125], *duplicate voting detection*, and *dispute-freeness*, which we discussed in Appendix B.2.

7 Related work

In this section, we compare and discuss VeriVoting with related electronic voting schemes designed to be decentralized, verifiable, and privacy-preserving, respectively. This is then followed by a qualitative summary in Table 6.

Decentralization. Decentralized voting systems have gradually drawn the attention of researchers over the last few years. McCorry et al.[114] proposed the first implementation of a decentralized and self-tallying voting scheme, which utilizes smart contracts to manage the entire voting process, and a two-round protocol (called Open Vote Network, OVN)[5] to make the scheme self-tallying. Hereafter, OVN with blockchain technology was used in several decentralized voting schemes[56-59] and [114]-[120]. However, a limitation of OVN is that the number of voters needs to be fixed before voting, which means these schemes do not support the dynamic join for voters. Also, the computational cost of this voting algorithm is related to the number of voters.

There are several decentralized voting schemes[61-65] with the linkable ring signatures technology [2,60]. In these schemes, to detect two signatures generated by the same voter, the verifier must execute the Link algorithm to check if each received signature is linked to an already stored signature, bringing an expensive computational cost to the verifier. New voters cannot dynamically participate in the vote after a fixed number of voters initialize the ring signature public keys. To avoid the problem of being unable to count votes due to the absence of voters, several schemes[66-70] use secret sharing technology [3, 71] to overcome the challenge. A drawback of these schemes is that if the authority responsible for distributing secret values is dishonest, the confidentiality of ballots will disappear.

Verifiability. As remarked in [72], if both the ciphertext and decryption key are verifiable, so does the corresponding decryption result. VeriVoting follows this rule to achieve verifiability of the tally result. Another widely used way to achieve this point is based on the verifiability of the ciphertext and decryption result, such as [73-79], and these schemes feature individual, universal [80,81] or E2E[82] verifiability. However, since they rely on one or more trustees (or called tellers/talliers) to tally the outcome of elections, an assumption is that at least one trustee must be honest; otherwise, the verifiability and privacy do not hold. Nevertheless, in our scheme, we leverage the fact that there is a competitive relationship between candidates to guarantee that at least one candidate will not collude with other candidates, so our assumption is weaker than that of the other schemes.

On the one hand, in the tally phase of these schemes, at least two statements should be proved: one is the correctness of the decryption for each ballot ciphertext, and the other is the completeness of the tally. On the other hand, the computational cost of the tally algorithm is related to the number of voters. These two factors will make large-scale impractical.

Privacy-preserving. In e-voting, privacy-preserving consists of ballot privacy(BP) and anonymity(Anon.), or at least one of two[83]. To keep ballot privacy, technologies like homomorphic encryption[84], threshold decryption [85], secret sharing, and mix-net[86] are employed [87-94]. To provide voters anonymity, there are schemes [95-100] adopting pseudonyms, blind signatures[101], group signatures [102], ring signatures, and NIZKs. Also, some schemes consider two properties together to achieve *bidirectional unlinkability*(BU) [95,103-105]. However, in these schemes, the public parameters, including encryption keys and common reference string, etc, are generated together by trustees or $\mathcal{EA}$, this implies that ballot privacy and anonymity can be compromised together if a collusion attack occurs. Instead, our scheme decouples ballot privacy and anonymity by generating public parameters separately from different parties.

For simplicity, we compared the above properties with previous works, as shown in Table 6.

8 Conclusion

In this paper, we highlight three essential properties that modern e-voting should have and propose the first weighted voting scheme that sufficiently meets the claimed properties of decentralization, verifiability, and privacy-preserving. Compared to previous works, our design offers a novel, practically relevant trade-off between security and performance for large-scale elections. With the computation verifiability feature of zkSNARKs, VeriVoting supports complex ballot formats, along with complex well-formedness conditions. Furthermore, VeriVoting, as a general e-voting framework, is compatible with other linear homomorphic encryption and NIZKs technologies. With the development of these technologies our scheme relies on, VeriVoting will arrive at a new balance point between security and performance.

Table 6: Comparison of different voting schemes

Schemes	Decentralization	Verifiability			Privacy-preserving			Cost of Tally alg.	Dynamic Join
		Ballot	DK	Decryption	BP	Anon.	BU		
[117]	✓	✓	×	N/A	✓	×	×	$O(\lvert\mathbb{V}\rvert)$	×
[115]	✓	✓	×	N/A	✓	×	×	$O(\lvert\mathbb{V}\rvert)$	×
[77]	×	✓	N/A	×	✓	×	×	$O(\lvert\mathbb{V}\rvert)$	✓
[73]	×	✓	N/A	✓	✓	×	×	$O(\lvert\mathbb{V}\rvert)$	✓
[103]	✓	×	N/A	×	✓	✓	×	$O(\lvert\mathbb{V}\rvert)$	×
[106]	×	✓	✓	N/A	✓	×	×	$O(\lvert\mathbb{C}\rvert)$	✓
VeriVoting	✓	✓	✓	N/A	✓	✓	✓	$O(\lvert\mathbb{C}\rvert)$	✓

✓ denotes satisfaction; × denotes dissatisfaction; N/A denotes no need.

ACKNOWLEDGMENTS

REFERENCES

[1]. Bin Yu, Joseph Liu, Amin Sakzad, Surya Nepal, Ron Steinfeld, Paul Rimba, and Man Ho Au. 2018. Platform-independent secure blockchain-based voting system. In Proceedings of the International Information Security Conference (ISC '18). Springer Press, Guildford, UK, 369-386. https://doi.org/10.1007/978-3-319-99136-8_20

[2]. Michael Backes, Nico Döttling, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. 2019. Ring signatures: logarithmic-size, no setup—from standard assumptions. In Proceedings of the EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '19). Springer Press, Darmstadt, Germany, 281-311. https://doi.org/10.1007/978-3-030-17659-4_10

[3]. Adi Shamir. 1979. How to share a secret. Communications of the ACM. ACM Press, New York, NY, United States, 1979, 612–613. https://doi.org/10.1145/359168.359176

[4]. Kim Ramchen, Chris Culnane, Olivier Pereira, and Vanessa Teague. 2019. Universally Verifiable MPC and IRV Ballot Counting. In Proceedings of the Financial Cryptography and Data Security: 23rd International Conference (FC '19). Springer-Verlag Press, Berlin, Heidelberg, 301–319. https://doi.org/10.1007/978-3-030-32101-7_19

[5]. Hao Feng, Peter YA Ryan, and Piotr Zieliński. 2010. Anonymous voting by two-round public discussion. IET Information Security . Wiley Press, Hoboken NJ, 2010, 62-67. https://doi.org/10.1049/iet-ifs.2008.0127

[6]. Thomas Bocek, and Burkhard Stiller. 2017. Smart contracts–blockchains in the wings. Digital marketplaces unleashed. Digital marketplaces unleashed. Springer Press, Berlin, Heidelberg, 2017, 169-184. https://doi.org/10.1007/978-3-662-49275-8_19

[7]. Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. 2020. An overview on smart contracts: Challenges, advances and platforms. Future Generation Computer Systems. Elsevier Press, 2020, 475-491. https://doi.org/10.1016/j.future.2019.12.019

[8]. Dalia Khader, Ben Smyth, Peter Ryan, and Feng Hao. 2012. A fair and robust voting system by broadcast. Lecture Notes in Informatics. Gesellschaft fur Informatik (GI) Press, 2012, 285-299. http://hdl.handle.net/10993/25419

[9]. Cortier Véronique, Galindo David, Küsters R Ralf, Müller Johannes, and Truderung Tomasz. 2016. SoK: Verifiability Notions for E-Voting Protocols. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (S&P '16). IEEE Press, San Jose, CA, USA, 779-798. http://doi.org/10.1109/SP.2016.52

[10]. Bernhard D, Cortier V, Galindo D, Pereira O, and Warinschi B. 2015. SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions. In Proceedings of the 2015 IEEE Symposium on Security and Privacy (S&P '15). IEEE Press, San Jose, CA, USA, 499-516. http://doi.org/10.1109/SP.2015.37

[11]. Malik Najmus Saqib, Junaid Kiani, Basit Shahzad, Adeel Anjum, Saif ur Rehman Malik, Naveed Ahmad, and Atta ur Rehman Khan. 2019. Anonymous and formally verified dual signature based online e-voting protocol. Cluster Computing. Springer Press, Berlin, Heidelberg, Germany, 2019, 1703–1716. https://doi.org/10.1007/s10586-018-2162-7

[12]. Jérémy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. 2018. Decentralized Multi-Client Functional Encryption for Inner Product. In Proceedings of the Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT '18). Springer Press, Brisbane, QLD, Australia, 703-732. https://doi.org/10.1007/978-3-030-03329-3_24

[13]. Jérémy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. 2020. Dynamic Decentralized Functional Encryption. In Proceedings of the Cryptology - CRYPTO 2020: 40th Annual lnternational Cryptology Conference (CRYPTO '20). Springer Press, Santa Barbara, CA, USA, 747-775. https://doi.org/10.1007/978-3-030-56784-2_25

[14]. Shweta Agrawal, Michael Clear, Ophir Frieder, Sanjam Garg, Adam O'Neill, and Justin Thaler. 2020. Ad hoc multi-input functional encryption. In Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS '20). Schloss Dagstuhl--Leibniz-Zentrum fuer Informatik Press, Dagstuhl, Germany, 1-41. https://doi.org/10.4230/LIPIcs.ITCS.2020.40

[15]. Dinh Duy Nguyen, Duong Hieu Phan, and David Pointcheval. 2023. Verifiable Multi-Client Functional Encryption for Inner Product. Cryptology ePrint Archive. IACR Press, 2023, 268. https://eprint.iacr.org/2023/268

[16]. Kazue Sako, and Joe Kilian. 1995. Receipt-Free Mix-Type Voting Scheme. In Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '95). Springer Press, Saint-Malo, France, 393–403. https://doi.org/10.1007/3-540-49264-X_32

[17]. Michanel Schläpfer, Rolf Haenni, Reto Koenig, and Oliver Spycher. 2012. Efficient Vote Authorization in Coercion-Resistant Internet Voting. In Proceedings of the E-Voting and Identity (Vote-ID '11). Springer Press, Tallinn, Estonia, 71–88. https://doi.org/10.1007/978-3-642-32747-6_5

[18]. Alshehri Ali, Mohamed Baza, Gautam Srivastava, Wahid Rajeh, Majed Alrowaily, and Majed Almusali. 2023. Privacy-Preserving E-Voting System Supporting Score Voting Using Blockchain. Applied Sciences Press, 2023, 1096. https://doi.org/10.3390/app13021096

[19]. Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In Proceedings of the IEEE Symposium on Security and Privacy (S&P '14). IEEE Press, Berkeley, CA, USA , 459–474. https://doi.org/10.1109/SP.2014.36

[20]. Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In Proceedings of the IEEE Symposium on Security and Privacy (S&P '16). IEEE Press, San Jose, CA, USA, 839-858. https://doi.org/10.1109/SP.2016.55

[21]. Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. 2020. ZEXE: Enabling Decentralized Private Computation. In Proceedings of the IEEE Symposium on Security and Privacy (S&P '20). IEEE Press, San Francisco, CA, USA, 947-964. https://doi.org/10.1109/SP40000.2020.00050

[22]. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2013. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Proceedings of the forty-fifth annual ACM symposium on Theory of Computing (STOC '13). ACM Press, New York, NY, USA, 111–120. https://doi.org/10.1145/2488608.2488623

[23]. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. 2019. Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19). ACM Press, New York, NY, USA, 2111–2128. https://doi.org/10.1145/3319535.3339817

[24]. Parno Bryan, Howell Jon, Gentry Craig, and Raykova Mariana. 2013. Pinocchio: Nearly Practical Verifiable Computation. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (S&P '13). IEEE Press, Berkeley, CA, USA, 2013, 238-252. https://doi.org/10.1109/SP.2013.47

[25]. Ethereum. Ethereum ide and tools for the web. [Online]. Available: http://remix.ethereum.org/

[26]. Jacob Eberhardt, and Stefan Tai. 2018. ZoKrates - Scalable Privacy-Preserving Off-Chain Computations. In Proceedings of the IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). IEEE Press, Halifax, NS, Canada, 1084-1091. https://doi.org/10.1109/Cybermatics_2018.2018.00199

[27]. Dodis Yevgeniy, Katz Jonathan, Xu Shouhuai, and Yung Moti. 2002. Key-Insulated Public Key Cryptosystems. In Proceedings of the EUROCRYPT: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam (EUROCRYPT '02). Springer Press, Amsterdam, The Netherlands, 65–82. https://doi.org/10.1007/3-540-46035-7_5

[28]. Jing Liu, and Bo Yang. 2011. Collusion-Resistant Multicast Key Distribution Based on Homomorphic One-Way Function Trees. IEEE Transactions on Information Forensics and Security. IEEE Press, CA, USA, 2011, 980-991. https://doi.org/10.1109/TIFS.2011.2144584

[29]. Jens Groth. 2010. Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In Proceedings of the16th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT '10). Springer Press, Singapore, 321–340. https://doi.org/10.1007/978-3-642-17373-8_19

[30]. Mihir Bellare, and Phillip Rogaway. 1993. Random oracles are practical: a paradigm for designing efficient protocols. In Proceedings of the 1st ACM conference on Computer and communications security (CCS '93). ACM Press, New York, NY, USA, 62–73. https://doi.org/10.1145/168588.168596

[31]. Jan Camenisch, Aggelos Kiayias, and Moti Yung. 2009. On the Portability of Generalized Schnorr Proofs. In Proceedings of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '09). Springer Press, Cologne, Germany, 425-442. https://doi.org/10.1007/978-3-642-01001-9_25

[32]. Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In Proceedings of the EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '16). Springer Press, Vienna, Austria, 305-326. https://doi.org/10.1007/978-3-662-49896-5_11

[33]. Jens Groth, and Mary Maller. 2017. Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs. In Proceedings of the Cryptology – CRYPTO 2017: 37th Annual International Cryptology Conference (CRYPTO '17). Springer Press, Santa Barbara, CA, USA, 581-612. https://doi.org/10.1007/978-3-319-63715-0_20

[34]. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS '12). ACM Press, New York, NY, USA, 326–349. https://doi.org/10.1145/2090236.2090263

[35]. Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin Vechev. 2022. ZeeStar: Private Smart Contracts by Homomorphic Encryption and Zero-knowledge Proofs. In Proceedings of the 2022 IEEE Symposium on Security and Privacy (S&P '22). IEEE Press, San Francisco, CA, USA, 179-197. https://doi.org/10.1109/SP46214.2022.9833732

[36]. Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. 2019. Zkay: Specifying and Enforcing Data Privacy in Smart Contracts. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19). ACM Press, New York, NY, USA, 1759–1776. https://doi.org/10.1145/3319535.3363222

[37]. Jens Groth, and Amit Sahai. 2008. Efficient Non-interactive Proof Systems for Bilinear Groups. In Proceedings of the Advances in Cryptology – EUROCRYPT 2008: 17th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '08). Springer Press, Berlin, Heidelberg, 415-532. https://doi.org/10.1007/978-3-540-78967-3_24

[38]. Shafi Goldwasser, and Yael Tauman Kalai. 2003. On the (In)security of the Fiat-Shamir Paradigm. In Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03). IEEE Computer Society Press, Cambridge, MA, USA, 102-113. http://doi.org/10.1109/SFCS.2003.1238185

[39]. Amos Fiat, and Adi Shamir. 1987. How to prove yourself: practical solutions to identification and signature problems. In Proceedings of the CRYPTO 1986: 6th Annual International Cryptology Conference (CRYPTO '86). Springer Press, Berlin, Heidelberg, 186–194. https://doi.org/10.1007/3-540-47721-7_12

[40]. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1986. How to construct random functions. ACM Press, New York, NY, USA, 1986, 792–807. https://doi.org/10.1145/6490.6503

[41]. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1985. On the Cryptographic Applications of Random Functions (Extended Abstract). In Proceedings of the CRYPTO 1984: 4th Annual lnternational Cryptology Conference (CRYPTO '84). Springer Press, Berlin, Heidelberg, 276-288. https://doi.org/10.1007/3-540-39568-7_22

[42]. Claude Crépeau, Commitment, Cryptography and Quantum Information Lab, McGill University School of Computer Science, accessed April 11, 2008

[43]. Mouris Dimitris, and Nektarios Georgios Tsoutsos. Masquerade: Verifiable multi-party aggregation with secure multiplicative commitments. Cryptology ePrint Archive.IACR Press, 2021. https://eprint.iacr.org/2021/1370.pdf

[44]. Thomas Bocek, and Burkhard Stiller. 2018. Smart Contracts – Blockchains in the Wings. Digital Marketplaces Unleashed. Springer Press, Berlin, Heidelberg, 2018, 169-184. https://doi.org/10.1007/978-3-662-49275-8_19

[45]. Sean Bowe, Ariel Gabizon, and Matthew D. Green. 2018. A Multi-party Protocol for Constructing the Public Parameters of the Pinocchio zk-SNARK. In Proceedings of the Financial Cryptography and Data Security (FC '18). Springer Press, Berlin, Heidelberg, 64–77. https://doi.org/10.1007/978-3-662-58820-8_5

[46]. INTEL CORP. Product Change Notification 114074-00. 2015. https://qdms.intel.com/dm/i.aspx/5A160770-FC47-47A0-BF8A-062540456F0A/PCN114074-00.pdf.

[47]. Beniiche Abdeljalil. 2020. A Study of Blockchain Oracles. https://doi.org/10.48550/arXiv.2004.07140

[48]. W Penard, and T van Werkhoven. On the secure hash algorithm family. Cryptogr. Context, 2008, 1–18

[49]. C Reitwiessner. 2021. Precompiled contracts for addition and scalar multiplication on the elliptic curve alt bn128. https://eips.ethereum.org/EIPS/eip-196.

[50]. Vikram Saraph, and Maurice Herlihy. 2019. An Empirical Study of Speculative Concurrency in Ethereum Smart Contracts. In Tokenomics (OASIcs '19). Schloss Dagstuhl, 1-15. https://doi.org/10.48550/arXiv.1901.01376

[51]. Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference (EuroSys '18). ACM Press, New York, NY, USA, 1-15. https://doi.org/10.1145/3190508.3190538

[52]. Denis Kirillov, Vladimir Korkhov, Vitaly Petrunin, Mikhail Makarov, Ildar M. Khamitov, and Victor Dostov. 2019. Implementation of an E-Voting Scheme Using Hyperledger Fabric Permissioned Blockchain. In Proceedings of the Computational Science and Its Applications (ICCSA '19). Springer Press, Saint Petersburg, Russia, 509-521. https://doi.org/10.1007/978-3-030-24296-1_40

[53]. Chaisawat S, and Vorakulpipat C. 2020. Fault-tolerant architecture design for blockchain-based electronics voting system. In Proceedings of the 2020 17th International Joint Conference on Computer Science and Software Engineering (JCSSE '20). IEEE Press, Bangkok, Thailand, 116-121. http://doi.org/10.1109/JCSSE49651.2020.9268264

[54]. Gongxian Zeng, Meiqi He, Siuming Yiu, and Zhengan Huang. 2021. Corrigendum to: A Self-Tallying Electronic Voting Based on Blockchain. The Computer Journal. Oxford University Press Press, United Kingdom, 2021, 3020–3034. https://doi.org/10.1093/comjnl/bxab123

[55]. Dimitriou Tassos. 2020. Efficient, coercion-free and universally verifiable blockchain-based voting. Computer Networks. Elsevier Press, Nx Amsterdam, Netherlands, 2020. https://doi.org/10.1016/j.comnet.2020.107234

[56]. Xuechao Yang, Xun Yi, Surya Nepal, Andrei Kelarev, and Fengling Han.2020. Blockchain voting: Publicly verifiable online voting protocol without trusted tallying authorities. Future Generation Computer Systems. Elsevier Press, Nx Amsterdam, Netherands, 2020, 859-874. https://doi.org/10.1016/j.future.2020.06.051

[57]. Tong Wu, Guomin Yang, Liehuang Zhu, and Yulin Wu. 2021. Privacy-preserving voluntary-tallying leader election for Internet of Things. Information Sciences. Elsevier Press New York, NY, USA, 2021, 461-472. https://doi.org/10.1016/j.ins.2021.06.028

[58]. Xuechao Yang, Xun Yi, Andrei Kelarev, Fengling Han, and Junwei Luo. 2021. A distributed networked system for secure publicly verifiable self-tallying online voting. Information Sciences. Elsevier Press, New York, NY, USA, 2021, 125-142. https://doi.org/10.1016/j.ins.2020.07.023

[59]. Dalia Khader, Ben Smyth, Peter Y. A. Ryan, and Feng Hao. 2012. A fair and robust voting system by broadcast. In Proceedings of the 5th International Conference on Electronic Voting 2012 (EVOTE '12). Gesellschaft fur Informatik (GI) press, Bonn, Germany, 285-299.https://subs.emis.de/LNI/Proceedings/Proceedings205/285.pdf

[60]. Joseph K. Liu, Man Ho Au, Willy Susilo, and Jianying Zhou. 2014. Linkable Ring Signature with Unconditional Anonymity. IEEE Transactions on Knowledge and Data Engineering. IEEE Press, Los Alamitos, USA, CA, 2014, 157-165. https://doi.org/10.1109/TKDE.2013.17

[61]. Alexandra Kugusheva, and Yury Yanovich. 2020. Ring Signature-Based Voting on Blockchain. In Proceedings of the 2020 2nd International Conference on Blockchain Technology and Applications (ICBTA '20). ACM Press, New York, NY, USA, 70–75. https://doi.org/10.1145/3376044.3376054

[62]. Lyu J, Jiang Z L, Wang X, et al. 2019. A secure decentralized trustless E-voting system based on smart contract. In Proceedings of the 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE '19). IEEE Press, Los Alamitos, CA USA, 570-577. https://doi.org/10.1109/TrustCom/BigDataSE.2019.0008

[63]. Deng Robert H, Bao Feng, Pang HweeHwa, and Zhou Jianying. 2005. Short linkable ring signatures for e-voting, e-cash and attestation. In Proceedings of the 1st Information Security Practice and Experience Conference (ISPEC '05). Springer Press, Berlin Heidelberg, 48-60. https://doi.org/10.1007/978-3-540-31979-5_5

[64]. Zhu Yan, Zichuan Zeng, and Chunli Lv. 2018. Anonymous voting scheme for boardroom with blockchain. International journal of performability engineering. RAMS Consultants, 2018, 2414-2422.https://doi.org/10.23940/ijpe.18.10.p17.24142422

[65]. Curran Kevin. 2018. E-Voting on the Blockchain. The Journal of the British Blockchain Association. The British Blockchain Association Press, United Kingdom,2018, 2516-3949. https://doi.org/10.31585/jbba-1-2-(3)2018

[66]. Li J, Wang X, Huang Z, Wang L, and Xiang Y. 2019. Multi-level multi-secret sharing scheme for decentralized e-voting in cloud computing. Journal of Parallel and Distributed Computing. Academic Press Inc Press, United States, 2019, 91-97. https://doi.org/10.1016/j.jpdc.2019.04.003

[67]. Liu Y, and Zhao Q. 2019. E-voting scheme using secret sharing and K-anonymity. In World Wide Web. Springer Press, Guildford, UK, 2019, 1657-1667. https://doi.org/10.1007/s11280-018-0575-0

[68]. Huang J, He D, Chen Y, Khan M K, and Luo M. 2022. A Blockchain-Based Self-Tallying Voting Protocol With Maximum Voter Privacy. IEEE Transactions on Network Science and Engineering, 3808-3820. IEEE Press, Los Alamitos, CA USA, 2022, 3808-3820 https://doi.org/10.1109/TNSE.2022.3190909

[69]. Bartolucci S, Bernat P, and Joseph D. 2018. SHARVOT: secret SHARe-based VOTing on the blockchain. In Proceedings of the 1st international workshop on emerging trends in software engineering for blockchain. IEEE Press, Gothenburg, SWEDEN, 30-34. https://doi.org/10.1145/3194113.3194118

[70]. Hsiao J H, Tso R, Chen C M, and Wu, M. E. 2018. Decentralized E-voting systems based on the blockchain technology. In Proceedings of the 2018 Computer Science and Ubiquitous Computing(CSA-CUTE '18). Springer Press,305-309. https://doi.org/10.1007/978-981-10-7605-3_50

[71]. Naor Moni, and Adi Shamir. 1995. Visual cryptography. In Proceedings of the 1995 EUROCRYPT: Workshop on the Theory and Application of Cryptographic Techniques Perugia(EUROCRYPT '95). Springer Press, Berlin Heidelberg, 1-12. https://doi.org/10.1007/BFb0053419

[72]. Saikrishna Badrinarayanan, Vipul Goyal, Aayush Jain, and Amit Sahai. 2016. Verifiable Functional Encryption. In Proceedings of the 2016 ASIACRYPT: Lecture Notes in Computer Science(ASIACRYPT '16). Springer Press, Berlin, Heidelberg, 557-587. https://doi.org/10.1007/978-3-662-53890-6_19.

[73]. Iovino Vincenzo, Rial Alfredo, Ronne Peter B, and Ryan Peter Y. A. 2020. (Universal)Unconditional Verifiability in E-Voting without Trusted Parties. In Proceedings of the 33th IEEE Computer Security Foundations Symposium (CSF '20). IEEE Press, New York, NY USA, 33-48. https://doi.org/10.1109/CSF49147.2020.00011

[74]. Sébastien Canard1, David Pointcheval, Quentin Santos, and Jacques Traoré. 2018. Practical strategy-resistant privacy-preserving elections. In Proceedings of the 2018 Computer Security: European Symposium on Research in Computer Security (ESORICS '18). Springer Press, Barcelona Spain, 331-349. https://doi.org/10.1007/978-3-319-98989-117

[75]. Xingyue Fan, Ting Wu, Qiuhua Zheng, Yuanfang Chen, Muhammad Alam, and Xiaodong Xiao. 2020. HSE-Voting: A secure high-efficiency electronic voting scheme based on homomorphic signcryption. Future Generation Computer Systems. Elsevier Press, Nx Amsterdam, Netherands, 2020, 754-762. https://doi.org/10.1016/j.future.2019.10.016

[76]. Kuesters Ralf, Liedtke Julian, Mueller Johannes, Rausch Daniel, and Vogt Andreas. 2020. Ordinos: a verifiable tally-hiding e-voting system.In Proceedings of the 2020 IEEE European Symposium on Security and Privacy (EuroS&P '20). IEEE Press, Los Alamitos, 216-235. https://doi.org/10.1109/EuroSP48549.2020.00022

[77]. Nicolas Huber, Ralf Küsters, Toomas Krips, Julian Liedtke, Johannes Müller, Daniel Rausch, Pascal Reisert, and Andreas Vogt. 2022. Kryvos: Publicly Tally-Hiding Verifiable E-Voting. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22). ACM Press, New York, NY, USA, 1443–1457. https://doi.org/10.1145/3548606.356yem0701

[78]. Boyen Xavier, Thomas Haines, and Johannes Müller. 2021. Epoque: practical end-to-end verifiable post-quantum-secure e-voting. In Proceedings of the IEEE European Symposium on Security and Privacy (EUROS&P '21). IEEE Press, New York, NY USA, 272-291. https://doi.org/10.1109/EuroSP51992.2021.00027

[79]. Chris Culnane, Peter Y. A. Ryan, Steve Schneider, and Vanessa Teague. 2015. VVote: A Verifiable Voting System. ACM Transactions on Information and System Security. ACM Press, New York, NY, USA, 2015, 1-30. https://doi.org/10.1145/2746338

[80]. Smyth Ben, Steven Frink, and Michael R. Clarkson.2015. Computational election verifiability: Definitions and an analysis of helios and JCJ. Technical Report Cryptology ePrint Archive. IACR Press, 2015. https://eprint.iacr.org/2015/233.

[81]. Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2010. Accountability: definition and relationship to verifiability. In Proceedings of the 17th ACM conference on Computer and communications security (CCS '10). ACM Press, New York, NY, USA, 526–535. https://doi.org/10.1145/1866307.1866366

[82]. Kiayias Aggelos, Thomas Zacharias, and Bingsheng Zhang. 2015. End-to-end verifiable elections in the standard model. In Proceedings of the 2015 EUROCRYPT: Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '15). Springer Berlin Heidelberg, 26-30. https://doi.org/10.1007/978-3-662-46803-6_16

[83]. Thomas Haines, Rafieh Mosaheb, Johannes Mueller, and Ivan Pryvalov. 2023. SoK: Secure e-voting with everlasting privacy. In Proceedings on Privacy Enhancing Technologies (PoPETs '23). IEEE Press, Lansanne, Switzerland, 279-293. https://doi.org/10.56553/popets-2023-0017

[84]. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In Proceedings of the Advances in Cryptology – ASIACRYPT 2017. ASIACRYPT 2017. Lecture Notes in Computer Science (LNCS '17). Springer Press, Cham, 409-437. https://doi.org/10.1007/978-3-319-70694-8_15

[85]. Berry Schoenmakers. 1999. A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting. In Proceedings of the Advances in Cryptology — CRYPTO' 99. CRYPTO 1999. Lecture Notes in Computer Science (LNCS '99). Springer Press, Berlin, Heidelberg, 148-164. https://doi.org/10.1007/3-540-48405-1_10

[86]. Markus Jakobsson, Ari Juels, and Ronald L. Rivest. 2002. Making mix nets robust for electronic voting by randomized partial checking. In Proceedings of the 11th USENIX Security Symposium (USS '02). USENIX Association Press, Berkeley, CA, USA, 339-353. https://eprint.iacr.org/2002/025

[87]. Johannes Buchmann, Denise Demirel, and Jeroen van de Graaf. 2013. Towards a Publicly-Verifiable Mix-Net Providing Everlasting Privacy. In Proceedings of the Financial Cryptography and Data Security. FC 2013. Lecture Notes in Computer Science (LNCS '13). Springer Press, Berlin, Heidelberg. 197-204. https://doi.org/10.1007/978-3-642-39884-1_16

[88]. Kristian Gjøsteen, Thomas Haines, and Morten Rotvold Solberg. 2021. Efficient Mixing of Arbitrary Ballots with Everlasting Privacy: How to Verifiably Mix the PPATC Scheme. In Proceedings of the Secure IT Systems. NordSec 2020. Lecture Notes in Computer Science (LNCS '21). Springer Press, Cham,92-107. https://doi.org/10.1007/978-3-030-70852-8_6

[89]. Tal Moran, and Moni Naor. 2010. Split-ballot voting: Everlasting privacy with distributed trust. ACM Transactions on Information and System Security. ACM Press, 2010, 246-255. https://doi.org/10.1145/1698750.1698756

[90]. Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. 2020. VOTEAGAIN: a scalable coercion-resistant voting system. In Proceedings of the 29th USENIX Conference on Security Symposium (SEC '20). USENIX Association Press, USA, 1553-1570. https://doi.org/10.48550/arXiv.2005.11189

[91]. Édouard Cuvelier, Olivier Pereira, and Thomas Peters. 2013. Election Verifiability or Ballot Privacy: Do We Need to Choose?. In Proceedings of the Computer Security – ESORICS 2013. ESORICS 2013. Lecture Notes in Computer Science (LSCS'13). Springer Press, Berlin, Heidelberg, 481-498. https://doi.org/10.1007/978-3-642-40203-6_27

[92]. Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. 2001. Sharing Decryption in the Context of Voting or Lotteries. In Proceedings of the Financial Cryptography. FC 2000. Lecture Notes in Computer Science (LNCS '01). Springer Press, Berlin, Heidelberg, 90-104. https://doi.org/10.1007/3-540-45472-1_7

[93]. Aram Jivanyan, and Araon Feickert. 2022. Aura: private voting with reduced trust on tallying authorities. Cryptology ePrint Archive. IACR Press, 2022. https://eprint.iacr.org/2022/543

[94]. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Mariya Izabachène. 2016. A Homomorphic LWE Based E-voting Scheme. In Proceedings of the Post-Quantum Cryptography. PQCrypto 2016. Lecture Notes in Computer Science (LNCS '16). Springer Press, Cham, 245-265. https://doi.org/10.1007/978-3-319-29360-8_16

[95]. Mahender Kumar, Satish Chand, and C.P. Katti. 2020. A Secure End-to-End Verifiable Internet-Voting System Using Identity-Based Blind Signature. IEEE Systems Journal. IEEE Press, 2020, 2032-2041. https://doi.org/10.1109/jsyst.2019.2940474

[96]. Mahender Kumar, C.P. Katti and P. C. Saxena. 2017. An Identity-based Blind Signature Approach for E-voting System. International Journal of Modern Education and Computer Science. MECS Press, 47-54. https://doi.org/10.5815/ijmecs.2017.10.06

[97]. Julio César Perez Carcia, A. Benslimane and S. Boutalbi. 2021. Blockchain-based system for e-voting using Blind Signature Protocol. In Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM '21). IEEE Press, Madrid, Spain, 01-06, https://doi.org/10.1109/GLOBECOM46510.2021.9685189

[98]. Lukas Malina, Jan Smrz, Jan Hajny, and Kamil Vrba. 2015. Secure electronic voting based on group signatures. In Proceedings of the 2015 38th International Conference on Telecommunications and Signal Processing (TSP '15). IEEE Press, Prague, Czech Republic, 06-10. https://doi.org/10.1109/tsp.2015.7296214

[99]. Ennan Zhai, David Isaac Wolinsky, Ruichuan Chen, Ewa Syta, Chao Teng, and Bryan Ford. 2016. AnonRep: towards tracking-resistant anonymous reputation. In Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation (NSDI '16). USENIX Association Press, USA, 583–596. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/zhai

[100]. Véronique Cortier, Constantin Cătălin Drăgan, François Dupressoir, Benedikt Schmidt, Pierre-Yves Strub, and Bogdan Warinschi. 2017. Machine-Checked Proofs of Privacy for Electronic Voting Protocols. In Proceedings of the IEEE Symposium on Security and Privacy (S&P '17). IEEE Press, San Jose, CA, USA, 993-1008. https://doi.org/10.1109/sp.2017.28

[101]. David Chaum. 1983. Blind signatures for untraceable payments. In Proceedings of the Advances in Cryptology: Proceedings of Crypto 82 (Crypto '82). New York, USA, Plenum Press, 199-203. https://doi.org/10.1007/978-1-4757-0602-4_18

[102]. Mihir Bellare, Haixia Shi, Chong Zhang. 2005. Foundations of Group Signatures: The Case of Dynamic Groups. In Proceedings of the 2005 Cryptographer's Track at the RSA Conference on Cryptographic Techniques and Security Practices. Lecture Notes in Computer Science (LNCS '05). Springer-Verlag Press, Berlin, Germany, 136-153. https://doi.org/10.1007/978-3-540-30574-3_11

[103]. Siqi Lu, Zhaoxuan Li, Xuyang Miao, Qingdi Han, and Jianhua Zheng. 2022. PIWS: Private Intersection Weighted Sum Protocol for Privacy-Preserving Score-Based Voting with Perfect Ballot Secrecy. IEEE Transactions on Computational Social Systems. IEEE Press, 2022, 1-18. https://doi.org/10.1109/TCSS.2022.3162869

[104]. Huilin Li, Yannan Li, Yong Yu, Baocang Wang, and Kefei Chen. 2021. A Blockchain-Based Traceable Self-Tallying E-Voting Protocol in AI Era. In *IEEE Transactions on Network Science and Engineering*. IEEE Press, 2021, 1019-1032, https://doi.org/10.1109/TNSE.2020.3011928

[105]. Vincenzo Agate, Alessandra De Paola, Pierluca Ferraro, Giuseppe Lo Re, and Marco Morana. 2021. SecureBallot: A secure open source e-Voting system. Journal of Network and Computer Applications. ACM Press, 2021, https://doi.org/10.1016/j.jnca.2021.103165

[106]. Xinyu Zhang, Bingsheng Zhang, Aggelos Kiayias, Thomas Zacharias, and K ui Ren. 2022. An Efficient E2E Crowd Verifiable E-Voting System. IEEE Tran sactions on Dependable and Secure Computing. IEEE Press, 3607-3620.https: //doi.org/10.1109/TDSC.2021.3103336

[107]. Sebastain Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele V enturi. 2012. On the Non-malleability of the Fiat-Shamir Transform. In Proc eedings of the 13th International Conference on Cryptology in India (INDO CRYPT '12). Springer Press, Kolkata, India, 60-79. https://doi.org/10.1007/978 -3-642-34931-7

[108]. Schnorr C. P.. 1991. Efficient signature generation by smart cards. Journal of Cryptology, 161-174. https://doi.org/10.1007/BF00196725

[109]. F. Hao. 2017. Schnorr Non-interactive Zero-Knowledge Proof. RFC Editor. T ech. Rep. 2070-1721. [Online]: https://www.rfc-editor.org/rfc/rfc8235

[110]. Stefano Tessaro, and David A. Wilson. 2014. Bounded-Collusion Identity-Ba sed Encryption from Semantically-Secure Public-Key Encryption: Generic C onstructions with Short Ciphertexts. In Proceedings of the 17th Internationa l Conference on Practice and Theory in Public-Key Cryptography (PKC '14). Springer Press, Buenos Aires, Argentina, 257-274. https://doi.org/10.1007/97 8-3-642-54631-0_15

[111]. T. Elgamal. 1985. A public key cryptosystem and a signature scheme based o n discrete logarithms. In Proceedings of the IEEE Transactions on Informati on Theory (IEEE T INFORM THEORY '85). IEEE Press, USA, 469-472. https: //doi.org/10.1109/TIT.1985.1057074

[112]. Yiannis Tsiounis, and Moti Yung. 1998. On the security of ElGamal-based en cryption. In Proceedings of the First International Workshop on Practice an d Theory in Public Key Cryptography (PKC '98). Springer Press, Pacifico Yo kohama, Japan, 117–134. https://doi.org/10.1007/BFb0054019

[113]. K. Kobara, and H. Imai. 2003. On the one-wayness against chosen-plaintext attacks of the Loidreau's modified McEliece PKC. In Proceedings of the IEEE Transactions on Information Theory (IEEE T INFORM THEORY '03). IEEE Press, USA, 3160-3168. https://doi.org/10.1109/TIT.2003.820016

[114]. McCorry Patrick, Shahandashti Siamak F, and Hao Feng. 2017. A smart cont ract for boardroom voting with maximum voter privacy. In Proceedings of t he International Conference on Financial Cryptography and Data Security (FC '17). Springer Press, Berlin, Germany, 357-375. https://doi.org/10.1007/97 8-3-319-70972-7_20

[115]. Yannan Li, Willy Susilo, Guomin Yang, Yong Yu, Dongxi Liu, Xiaojiang Du, and Mohsen Guizani. 2022. A Blockchain-Based Self-Tallying Voting Protoc ol in Decentralized IoT. IEEE Transactions on Dependable and Secure Comp uting. IEEE Press, Los Alamitos, 2022, 119-130. https://doi.org/10.1109/TDS C.2020.2979856

[116]. Panja Somnath, Bag Samiran, Hao Feng, and Roy Bimal. 2020. A Smart Cont ract System for Decentralized Borda Count Voting. IEEE Transactions on En gineering Management. IEEE Press, Piscataway, NJ, 2020, 1323-1339. https:// doi.org/10.1109/TEM.2020.2986371

[117]. Yang Yang, Zhangshuang Guan, Zhiguo Wan, Jian Weng, Hwee Hwa Pang, and Robert H. Deng. 2021. PriScore: Blockchain-Based Self-Tallying Election System Supporting Score Voting. IEEE Transactions on Information Forensi cs and Security. IEEE Press, Piscataway, NJ, 2021, 4705-4720. https://doi.org/ 10.1109/TIFS.2021.3108494

[118]. Gang Han, Yannan Li, Yong Yu, Kim-Kwang Raymond Choo, and Nadra Gui zani. 2020. Blockchain-Based Self-Tallying Voting System with Software Up dates in Decentralized IoT. IEEE Network. IEEE Press, Piscataway, NJ, 2020, 166-172. https://doi.org/10.1109/MNET.001.1900439

[119]. Christian Killer, Bruno Rodrigues, Eder John Scheid, Muriel Franco, Moritz Eck, Nik Zaugg, Alex Scheitlin, and Burkhard Stiller. 2020. Provotum: A Blo ckchain-based and End-to-end Verifiable Remote Electronic Voting System I n Proceedings of the Annual IEEE Conference on Local Computer Networks (LCN '20). IEEE Press, Los Alamitos, CA, USA, 172-183. https://doi.org/10.11 09/LCN48667.2020.9314815

[120]. Huang J, He D, Chen Y, Khan M K, and Luo M. 2022. A Blockchain-Based Se lf-Tallying Voting Protocol With Maximum Voter Privacy. IEEE Transaction s on Network Science and Engineering, 3808-3820. IEEE Press, Los Alamitos, CA USA, 2022, 3808-3820 https://doi.org/10.1109/TNSE.2022.3190909

[121]. Morio Kevin,and Künnemann Robert. 2021. Verifying Accountability for Un bounded Sets of Participants. In Proceedings of the IEEE Computer Security Foundations Symposium (CSF '21). IEEE Press, Los Alamitos, CA USA, 173-1 88. https://doi.org/10.1109/CSF51468.2021.00032

[122]. Stefano Tessaro, and David A. Wilson. 2014. Bounded-Collusion Identity-Ba sed Encryption from Semantically-Secure Public-Key Encryption: Generic C onstructions with Short Ciphertexts. In Proceedings of the 17th Internationa l Conference on Practice and Theory in Public-Key Cryptography (PKC 201 4). Springer Press, Buenos Aires, Argentina, 257-274. https://doi.org/10.1007/ 978-3-642-54631-0_15

[123]. Sato Shingo, and Junji Shikata. 2023. Compact Bounded-Collusion Identity-b ased Encryption via Group Testing. Cryptology ePrint Archive, IACR Press, 2023. https://eprint.iacr.org/2023/444

[124]. Shafi Goldwasser, Allison Lewko, and David A Wilson. 2012. Bounded-collu sion IBE from key homomorphism. In Proceedings of the Theory of Cryptog raphy Conference (TCC '12). Springer Press, Berlin, Heidelberg, 564-581. htt ps://doi.org/10.1007/978-3-642-28914-9_32

[125]. Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. 2 016. BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme. I n ACM CCS 2016. 1614–1625. https://doi.org/10.1145/2976749.2978337

[126]. Thomas Ristenpart and Scott Yilek. 2007. The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks. In EUROCRYPT 2007. Springer Press, volume 4515 of LNCS, pages 228–245. https://doi.org/1 0.1007/978-3-540-72540-4_13

[127]. WhiteHat, B., Baylina, J., & Bellés, M. (2020). Baby Jubjub elliptic curve. Ethe reum Improvement Proposal, EIP-2494, 29. https://doi.org/10.17487/RFC7748

APPENDIX A

A.1 CORRECTNESS

Correctness. By the LEH, LCH, and LKH properties, the correctness of our construction is straightforward.

$$\mathsf{Tally}(\mathbb{DK},\mathbb{CT}.\mathbb{C}_1)=\left\{\mathcal{E}.\mathsf{Dec}\left(DK_j, CT_j^1\right)\right\}_{j\in[\mathbb{C}]}$$

$$=\left\{\mathcal{E}.\mathsf{Dec}\left(\begin{array}{c}\prod_{j'}^{[\mathbb{C}]}\mathsf{KeyGen}\left(sk_{j'},CT_j^0\right),\\ \prod_{i}^{[\mathbb{V}]}\mathsf{Enc}\left(ek,\vec{v}_{i,j}\cdot w_i;r_{i,j}\right)\end{array}\right)\right\}_{j\in[\mathbb{C}]}$$

$$=\left\{\mathcal{E}.\mathsf{Dec}\left(\begin{array}{c}\mathsf{KeyGen}\left(\sum_{j'}^{[\mathbb{C}]}sk_{j'},\mathsf{Comm}\left(\sum_{i}^{[\mathbb{V}]}r_{i,j}\right)\right),\\ \mathsf{Enc}\left(\prod_{j'}^{[\mathbb{C}]}pk_{j'},\sum_{i=1}^{|\mathbb{V}|}\vec{v}_{i,j}\cdot w_i;\sum_{i=1}^{|\mathbb{V}|}r_{i,j}\right)\end{array}\right)\right\}_{j\in[\mathbb{C}]}\quad(\mathsf{LCH},\mathsf{LEH})$$

$$=\left\{\mathcal{E}.\mathsf{Decrypt}\left(\sum_{j'}^{[\mathbb{C}]}sk_{j'},\mathcal{E}.\mathsf{Encrypt}\left(\begin{array}{c}\prod_{j'}^{[\mathbb{C}]}pk_{j'},\sum_{i=1}^{|\mathbb{V}|}\vec{v}_{i,j}\cdot\\ w_i;\sum_{i=1}^{|\mathbb{V}|}r_{i,j}\end{array}\right)\right)\right\}\quad(\mathsf{LKH})$$

$$=\left\{\sum_{i=1}^{|\mathbb{V}|}\vec{v}_{i,j}\cdot w_i\right\}_{j\in[\mathbb{C}]}$$

A.2 INSTANTIATION

A.2.1 Building blocks

Fiat-Shamir based Non-Interactive Zero-Knowledge Argument of Knowledge (FS-NIZK[107,108]**)**. To efficiently prove the equality of discrete logarithms of decryption keys and public key generated by the same candidate, we use the FS-NIZK argument systems to achieve this. The FS-NIZK argument systems meet simulation soundness and simulation extraction in the random oracle model[31,109]. Moreover, the FS-NIZK argument systems are always more efficient than other argument/proof systems under the common reference string model.

BC-IND-Secure Encryption. As discussed in Section 2.3, we explain the security against bounded-collusion in our scheme, which is different from traditional bounded-collusion in IBE. Tessaro et al[110] presented an efficient scheme satisfying the bounded-collusion based on the ElGamal encryption scheme [111,112]. We take the modified ElGamal scheme as our BC-IND-secure encryption scheme in our instantiation.

A.2.2 Instantiation

• $\mathsf{Setup}(1^\lambda)$: Taking as input a security parameter $\boldsymbol{\lambda}$, and generates public parameters $pp_{sv}:=(\mathbb{G},p,g,h,\mathsf{Hash})$, where $\mathbb{G}$ is a cyclic group of prime order p, g is a generator of $\mathbb{G}$, $h\leftarrow_R\mathbb{G}$, Hash is a hash function that $\mathsf{Hash}:\{0,1\}^*\to\mathbb{Z}_{\boldsymbol{p}}$, which plays the random oracle in Fiat-Shamir based NIZK argument system $\Pi_{\mathsf{FS-NIZK}}$.

• $\mathsf{KeyGen}(pp_{sv})$: Taking as input the public parameters pp_{sv}, each candidate $\mathbb{c}_j\in\mathbb{C}$ generates a secret key $sk_j\leftarrow_R\mathbb{Z}_p$ as well as public key $\left(pk_j\leftarrow g^{sk_j}\right)\in\mathbb{G}$.

• $\mathsf{Vote}\left(pp_{sv},\{pk_j\}_{j\in[\mathbb{C}]},\vec{v}_i,w_i\right)$: Each voter $\mathbb{v}_i$ takes the following steps to encrypt its $\vec{v}_i$:

(1).Compute the aggregated encryption key $ek:=\prod_j^{[\mathbb{C}]}pk_j=g^{\sum_j^{[\mathbb{C}]}sk_j}\in\mathbb{G}$; (This step can be precomputed)

(2).For each $j\in[\mathbb{C}]$, compute ciphertexts $\left\{\left(ct_{i,j}^0\leftarrow g^{r_{i,j}},ct_{i,j}^1\leftarrow ek^{r_{i,j}}\cdot g^{v_{i,j}\cdot w_i}\right)\right\}_{j\in[\mathbb{C}]}$, where $r_{i,j}\leftarrow_R\mathbb{Z}_p$;

(3).Output $\vec{ct}_i:=\left\{\left(ct_{i,j}^0,ct_{i,j}^1\right)\right\}_{j\in[\mathbb{C}]}$.

▪ $\mathsf{VoteAgg}(\mathbb{CT}',\vec{ct}_i)$: Upon receiving the ballot ciphertexts $\vec{ct}_i$ from a voter $\mathbb{v}_i$, the aggregator $\mathbb{a}$ updates its state variable set $\mathbb{CT}=\left\{\left(\mathbb{CT}.CT_j^0=\mathbb{CT}'.CT_j^0\cdot ct_{i,j}^0,\mathbb{CT}.CT_j^1=\mathbb{CT}'.CT_j^1\cdot ct_{i,j}^1\right)\right\}_{j\in[\mathbb{C}]}$.

At the end of the voting phase, we have:

$$\mathbb{CT}=\left(\mathbb{CT}.\mathbb{C}_0=\left\{CT_j^0=g^{\sum_i^{[\mathbb{V}]}r_{i,j}}\right\}_{j\in[\mathbb{C}]},\mathbb{CT}.\mathbb{C}_1=\left\{CT_j^1=ek^{\sum_i^{[\mathbb{V}]}r_{i,j}}\cdot g^{\sum_i^{[\mathbb{V}]}v_{i,j}\cdot w_i}\right\}_{j\in[\mathbb{C}]}\right)\in(\mathbb{G}\times\mathbb{G})^{|\mathbb{C}|}.$$

• $\mathsf{DKeyGen}(sk_j,\mathbb{CT}.\mathbb{C}_0)$: When the voting phase finished, each candidate $\mathbb{c}_j$ generates the partial decryption keys $\overrightarrow{dk_j}:=\left\{dk_{j,j'}=\left(CT_{j'}^0\right)^{sk_j}\right\}_{j'\in[\mathbb{C}]}$ and corresponding NIZK argument of knowledge(AoK) to $\mathbb{a}$:

$$\Pi_{\mathsf{FS-NIZK}}.\mathsf{Prove}\left(pp_{sv},\overrightarrow{dk_j},sk_j\right)\to\pi_j:=\mathsf{AoK}\left\{\left(sk_j\right):\left(\log_{CT_{j'}^0}dk_{j,j'}=\log_g pk_j\right)_{j'\in[\mathbb{C}]}\right\}$$

▪ $\mathsf{DKeyAgg}(\overrightarrow{dk_j},\pi_j,\mathbb{DK}',\mathbb{CT}.\mathbb{C}_0)$: After receiving $\overrightarrow{dk_j}$ from candidate $\mathbb{c}_j\in\mathbb{C}$, $\mathbb{a}$ checks whether $\Pi_{\mathsf{FS-NIZK}}.\mathsf{Verify}\left(\overrightarrow{dk_j},\pi_j\right)=1$ If all positive, update the decryption key set $\mathbb{DK}=\left\{DK_{j'}=\mathbb{DK}'.DK_{j'}\cdot dk_j\right\}_{j'\in[\mathbb{C}]}$; else return $\perp$. Finally,

$$\mathbb{DK}=\left\{DK_j=\prod_{j'}^{[\mathbb{C}]}g^{sk_{j'}\cdot\sum_i^{[\mathbb{V}]}r_{i,j}}=g^{\sum_{j'}^{[\mathbb{C}]}sk_{j'}\cdot\sum_i^{[\mathbb{V}]}r_{i,j}}\right\}_{j\in[\mathbb{C}]}\in\mathbb{G}^{|\mathbb{C}|}$$

• $\mathsf{Tally}(\mathbb{DK},\mathbb{CT}.\mathbb{C}_1)\to\{t_j\}_{j\in[\mathbb{C}]}$: Takes as input the decryption key set $\mathbb{DK}$ and the aggregated ballot ciphertext set $\mathbb{CT}.\mathbb{C}_1$, outputs final tallies $\left\{t_j=\mathcal{E}.\mathsf{Dec}\left(DK_j,CT_j^1\right)=<\vec{v}_j,\vec{w}>\right\}_{j\in[\mathbb{C}]}\in\mathbb{Z}_p^{|\mathbb{C}|}$.

Correctness. The correctness is straightforward.

$$\begin{aligned}t_j&=\mathcal{E}.\mathsf{Dec}\left(DK_j,CT_j^1\right)\\&=CT_j^1/DK_j\\&=\frac{ek^{\sum_i^{[\mathbb{V}]}r_{i,j}}}{g^{\sum_{j'}^{[\mathbb{C}]}sk_{j'}\cdot\sum_i^{[\mathbb{V}]}r_{i,j}}}\cdot g^{\sum_i^{[\mathbb{V}]}v_{i,j}\cdot w_i}\\&=\frac{\left(g^{\sum_{j'}^{[\mathbb{C}]}sk_{j'}}\right)^{\sum_i^{[\mathbb{V}]}r_{i,j}}}{g^{\sum_{j'}^{[\mathbb{C}]}sk_{j'}\cdot\sum_i^{[\mathbb{V}]}r_{i,j}}}\cdot g^{\sum_i^{[\mathbb{V}]}v_{i,j}\cdot w_i}\\&=g^{\sum_i^{[\mathbb{V}]}v_{i,j}\cdot w_i}=g^{<\vec{v}_j,\vec{w}>}\end{aligned}$$

A.3 PROOF OF THEOREM 1.

Proof. We now show that under the assumption of computationally simulation-knowledge-sound NIZKs, BC-IND-secure encryption schemes, then for every real-world adversary $\mathcal{A}$ against our construction Π_{SV} there exists an ideal-world simulator $\mathcal{S}$ such that the view of the real-world and ideal-world adversaries are computationally indistinguishable. We will now describe the operation of the ideal-world simulator $\mathcal{S}$, which runs $\mathcal{A}$ internally and interacts with the ideal functionality $\mathcal{F}_{\mathrm{SV}}$.

We begin by sketching the operation of the ideal-world simulator $\mathcal{S}$.

There are three kinds of parties: aggregator, voter and candidate, where the aggregator is simulated by the simulator $\mathcal{S}$, since all messages to the aggregator functionality are public, simulating the aggregator functionality is trivial.

Malicious candidates and semi-honest voters are both corrupted. Before simulating the view of candidates and voters, we introduce an approach to making the tally outcome consistent in both worlds. Instead of uncorrupted candidates, the simulator $\mathcal{S}$ generates elaborate decryption keys to achieve the consistency requirement. Therefore, in the decryption key generation phase, the simulator $\mathcal{S}$ does the following:

1). By the simulation sound extractability of NIZK argument systems, extract secret keys $\{sk_j\}_{j\in[\mathbb{C}_c]}$ of all corrupted candidates in $\mathbb{C}_c$.

2). Decrypt all ciphertexts from voters in $\mathbb{V}$ to compute the ideal world's tally result $\{t'_j\}_{j\in[\mathbb{C}]}$ in advance.

3). According to the tally result $\{t_j\}_{j\in[\mathbb{C}]}$ from the real world, generate elaborate decryption keys of uncorrupted candidates: By LCH, LEH and LKH properties, the encryption key is $ek \coloneqq \prod_j^{[\mathbb{C}]} pk_j = \left[\sum_j^{[\mathbb{C}_c]} sk_j + \sum_j^{[\mathbb{C}\setminus\mathbb{C}_c]} sk_j\right]$, where we denote some mathematical operation by $'[\,]'$, such as modular exponentiation operation, $\{sk_j\}_{j\in[\mathbb{C}\setminus\mathbb{C}_c]}$ is randomly chosen by the simulator $\mathcal{S}$. The candidate $\mathbb{c}_j$'s aggregated ciphertext is $\left[\left(\sum_{j'}^{[\mathbb{C}_c]} sk_{j'} + \sum_{j'}^{[\mathbb{C}\setminus\mathbb{C}_c]} sk_{j'}\right)\cdot\sum_i^{[\mathbb{V}]} r_{i,j} + t'_j\right]$ where $t'_j = \sum_i^{[\mathbb{V}]} \tilde{v}_{i,j}\cdot\widetilde{w}_i$ is $\mathbb{c}_j$'s tally result in the ideal world. Then the simulator $\mathcal{S}$ defines a function: $f(u_j) \coloneqq \left(\left(\sum_{j'}^{[\mathbb{C}_c]} sk_{j'} + \sum_{j'}^{[\mathbb{C}\setminus\mathbb{C}_c]} sk_{j'}\right)\cdot\sum_i^{[\mathbb{V}]} r_{i,j}\cdot t_j^{-1} - \left(\sum_i^{[\mathbb{V}]} r_{i,j}\cdot\sum_{j'}^{[\mathbb{C}_c]} sk_{j'}\right) + t'_j\cdot t_j^{-1}\right)\cdot u_j$ and selects $\{u_j\}_{j\in[\mathbb{C}\setminus\mathbb{C}_c]}$ making $\sum_j^{[\mathbb{C}\setminus\mathbb{C}_c]} u_j \equiv 1$, generates partial decryption keys $\{dk_{j,j'} \coloneqq [f(u_{j'})]\}_{j'\in[\mathbb{C}\setminus\mathbb{C}_c]}$ for the candidate $\mathbb{c}_j$. In the tally phase, the correctness of decryption is as follows: (The other candidates' partial decryption keys can be created in the same way.)

$$\frac{CT_j^1}{DK_j} = \frac{\left[\left(\sum_{j'}^{[\mathbb{C}_c]} sk_{j'} + \sum_{j'}^{[\mathbb{C}\setminus\mathbb{C}_c]} sk_{j'}\right)\cdot\sum_i^{[\mathbb{V}]} r_{i,j} + t'_j\right]}{\left[\sum_i^{[\mathbb{V}]} r_{i,j}\cdot\sum_{j'}^{[\mathbb{C}_c]} sk_{j'} + \sum_{j'}^{[\mathbb{C}\setminus\mathbb{C}_c]} f(u_{j'})\right]}$$

$$= \frac{\left[\left(\sum_{j'}^{[\mathbb{C}_c]} sk_{j'} + \sum_{j'}^{[\mathbb{C}\setminus\mathbb{C}_c]} sk_{j'}\right)\cdot\sum_i^{[\mathbb{V}]} r_{i,j} + t'_j\right]}{\left[\begin{matrix}\sum_i^{[\mathbb{V}]} r_{i,j}\cdot\sum_{j'}^{[\mathbb{C}_c]} sk_{j'} + \left(\sum_{j'}^{[\mathbb{C}_c]} sk_{j'} + \sum_{j'}^{[\mathbb{C}\setminus\mathbb{C}_c]} sk_{j'}\right)\cdot\sum_i^{[\mathbb{V}]} r_{i,j}\cdot t_j^{-1} \\ -\left(\sum_i^{[\mathbb{V}]} r_{i,j}\cdot\sum_{j'}^{[\mathbb{C}_c]} sk_{j'}\right) + t'_j\cdot t_j^{-1}\end{matrix}\right]}$$

$$= \frac{\left[\left(\sum_{j'}^{[\mathbb{C}_c]} sk_{j'} + \sum_{j'}^{[\mathbb{C}\setminus\mathbb{C}_c]} sk_{j'}\right)\cdot\sum_i^{[\mathbb{V}]} r_{i,j} + t'_j\right]}{\left[\left(\left(\sum_{j'}^{[\mathbb{C}_c]} sk_{j'} + \sum_{j'}^{[\mathbb{C}\setminus\mathbb{C}_c]} sk_{j'}\right)\cdot\sum_i^{[\mathbb{V}]} r_{i,j}\cdot + t'_j\right)\cdot t_j^{-1}\right]} = [t_j]$$

To illustrate the point, we take the instantiation in Appendix A.2 as an example.

According to the instantiation, the aggregated ciphertext is $ek^{\sum_i^{[\mathbb{V}]} r_{i,j}}\cdot g^{\sum_i^{[\mathbb{V}]}\tilde{v}_{i,j}\cdot\widetilde{w}_i} = g^{\left(\sum_j^{[\mathbb{C}c]} sk_j + \sum_j^{[\mathbb{C}\setminus\mathbb{C}c]} sk_j\right)\cdot\sum_i^{[\mathbb{V}]} r_{i,j}}\cdot g^{\sum_i^{[\mathbb{V}]}\tilde{v}_{i,j}\cdot\widetilde{w}_i} = g^{\left(\sum_{j'}^{[\mathbb{C}c]} sk_{j'} + \sum_{j'}^{[\mathbb{C}\setminus\mathbb{C}c]} sk_{j'}\right)\cdot\sum_i^{[\mathbb{V}]} r_{i,j} + t'_j}$. Due to the simulator $\mathcal{S}$ knows all the secret keys $\{sk_j\}_{j\in[\mathbb{C}]}$, creating the decryption keys $\left\{dk_{j,j'} \coloneqq g^{f(u_{j'})}\right\}_{j'\in[\mathbb{C}\setminus\mathbb{C}_c]}$ with $CT_j^0 = g^{\sum_i^{[\mathbb{V}]} r_{i,j}}$ is trivial. The correctness of decryption is straightforward.

Next, according to our construction, the view of each malicious candidate $\mathbb{c}_j \in \mathbb{C}_c$ ($|\mathbb{C}_c| \le \delta_c$) and semi-honest voter $\mathbb{v}_i \in \mathbb{V}_c$ ($|\mathbb{V}_c| \le \delta_v$)in the real world is as follows:

$\mathrm{REAL}^{\Pi_{\mathrm{SV}}}_{\mathcal{A}_{\mathbb{V},\mathbb{C}}(z)}\left(\{\mathbb{v}_i\colon(\vec{v}_i, w_i)\}_{i\in[\mathbb{V}_c]}, \{\mathbb{c}_j\colon sk_j\}_{j\in[\mathbb{C}_c]}, \mathbb{a}\colon\bot\right)$
$\coloneqq \{\{(\vec{v}_i, w_i)\}_{i\in[\mathbb{V}_c]}, \{pk_j\}_{j\in[\mathbb{C}]}, pp_{sv}, \{\vec{ct}_i\}_{i\in[\mathbb{V}]}, \{\vec{dk}_j, \pi_j\}_{j\in[\mathbb{C}]}, \{t_j\}_{j\in[\mathbb{C}]}\}$

To simulate the view of the candidate and voter, the simulator $\mathcal{S}_{\mathbb{C}}$ does the following:

1) Run the $\mathrm{SV.Setup}(1^\lambda) \to \widetilde{pp_{sv}}$ containing $(\widetilde{crs}, \tau) \leftarrow \Pi_{\mathrm{NIZK}}.\mathrm{Setup}_{\mathrm{sim}}(1^\lambda)$, and initialize two sets $\widetilde{\mathbb{CT}}$ and $\widetilde{\mathbb{DK}}$, and a list L_v for storing secret ballots from $\mathbb{v}_i \in \mathbb{V}_c$.
Generate random public keys for uncorrupted candidates $\{\widetilde{pk}_j\}_{j\in[\mathbb{C}\setminus\mathbb{C}_c]}$ by random secret keys $\{\widetilde{sk}_j\}_{j\in[\mathbb{C}\setminus\mathbb{C}_c]}$.

2) On receiving $\vec{ct}_i$ from the voter in $\mathbb{V}_c$, record them into L_v, then update $\widetilde{\mathbb{CT}}$.

3) Randomly select bogus ballot ciphertexts from ciphertext space $\mathcal{C}$, output $\left\{\widetilde{\vec{ct}}_i\right\}_{i\in[\mathbb{V}\setminus\mathbb{V}_c]}$ and update $\widetilde{\mathbb{CT}}$.

4) On receiving $\left(\vec{dk}_j, \pi_j\right)$ from the candidate in $\mathbb{C}_c$, rewind and extract the knowledge of sk_j, then update $\widetilde{\mathbb{DK}}$.

5) If the candidate is uncorrupted, generate $\widetilde{\vec{dk}}_j$ by the above-described approach, run the zero-knowledge simulator to simulate a proof $\tilde{\pi}_j \leftarrow \Pi_{\mathrm{NIZK}}.\mathrm{Prove}_{\mathrm{sim}}(\widetilde{crs}, \tau, \widetilde{\vec{dk}}_j)$, then update $\widetilde{\mathbb{DK}}$.

6) The subsequent process is consistent with the construction.

We now show that the simulator's view is indistinguishable from that of the adversary via a series of hybrids. In both views, the tally result part is identical.

$\mathbf{H}_0$: This is the real world.

$\mathbf{H}_1$: In this hybrid, we run the knowledge extractor when encountering the output of any corrupted candidates $\left(\vec{dk}_j, \pi_j\right)$, and abort if the knowledge extractor fails. By Lemma A.1 we show that if the proof extractor fails with negligible probability, then $\mathrm{H}_0 \overset{\mathrm{c}}{\approx} \mathrm{H}_1$.

$\mathbf{H}_2$: In this hybrid, we replace all proofs $\{\vec{dk}_j, \pi_j\}_{j\in[\mathbb{C}_H]}$ by uncorrupted candidates with simulated proofs $\left\{\widetilde{\vec{dk}}_j, \tilde{\pi}_j\right\}_{j\in[\mathbb{C}_H]}$. By Lemma A.2, if the argument system is computational zero-knowledge, then $\mathrm{H}_1 \overset{\mathrm{c}}{\approx} \mathrm{H}_2$.

H$_3$: In this hybrid, we replace ciphertexts $\{\overrightarrow{ct_i}\}_{i\in[\mathbb{V}\backslash\mathbb{V}_c]}$ with $\{\widetilde{\overrightarrow{ct_i}}\}_{i\in[\mathbb{V}\backslash\mathbb{V}_c]}$. By Lemma A.3, if the encryption scheme is IND-secure, then $\mathrm{H}_2 \overset{c}{\approx} \mathrm{H}_3$.

H$_4$: The hybrid is the same as H_3 except for the following collusion attack. Malicious candidates in $\mathbb{C}\backslash\mathbb{C}_H$ collude together to try to decrypt a secret ballot $\overrightarrow{ct_{i^*}} = \{(ct^0_{i^*,j}, ct^1_{i^*,j})\}_{j\in[\mathbb{C}]}$ output by a voter $\mathbb{v}_{i^*}$. Firstly, we observe that the encryption key ek derives from the aggregation of public keys of all candidates in $\mathbb{C}$. Then, according to the LKH property, to decrypt the target ciphertext $\overrightarrow{ct_{i^*}}$, the collusive candidates need to generate partial decryption keys of all the candidates in $\mathbb{C}$ for $\{ct^0_{i^*,j}\}_{j\in[\mathbb{C}]}$, and combined them together as the valid decryption keys for $\{ct^1_{i^*,j}\}_{j\in[\mathbb{C}]}$. By Lemma A.4, if the encryption scheme is secure against bounded collusion and $|\mathbb{C}_H| \geq 1$, then $\mathrm{H}_3 \overset{c}{\approx} \mathrm{H}_4$.

H$_5$: The hybrid is the same as H_5 except for the following collusion attack. Semi-honest voters in $\mathbb{V}_c$ collude together to try to learn the content of an encrypted target ballot $\overrightarrow{ct_{i^*}} = \{(ct^0_{i^*,j}, ct^1_{i^*,j})\}_{j\in[\mathbb{C}]}$ from final tally result. We assumed that $|\mathbb{V}_H| = |\mathbb{V}\backslash\mathbb{V}_c| \geq 2$ and at least two voters with the same weight cast their ballots for different candidates. By Lemma A.7, if the encryption scheme is BC-IND-secure and $|\mathbb{V}_H| \geq 2$, then $\mathrm{H}_3 \overset{c}{\approx} \mathrm{H}_4$.

Note that Hybrid 5 is identical to the simulation. By summation over the previous hybrids we show that $\mathrm{H}_0 \overset{c}{\approx} \mathrm{H}_5$. We conclude our proof sketch by presenting the supporting lemmas.

Lemma A.1 *For any PPT adversary $\mathcal{A}$, if simulation sound extractable NIZK argument systems exist, then advantage $|adv^{\mathcal{A}}_{H_1} - adv^{\mathcal{A}}_{H_0}| \leq \epsilon$ where ϵ is the extraction failure probability.*

Proof. The simulator operates in the same manner, but we now extract when given $(\overrightarrow{dk_j}, \pi_j)$ of corrupted candidates who want to generate false decryption keys, i.e., invalid statements, to change the tally result. By definition, the extractor will fail with at most negligible probability ϵ because it deals solely with NIZKs, which have efficient extractors. Therefore, our proofs have knowledge extractors that succeed with probability 1 – negl(λ).

Lemma A.2 *For any PPT adversaries $\mathcal{A}$, if computational zero-knowledge NIZK argument systems exist, then the advantage $|adv^{\mathcal{A}}_{H_2} - adv^{\mathcal{A}}_{H_1}| \leq \epsilon$ where ϵ is the simulation failure probability.*

Proof. The simulator operates in the same manner, but we now simulate proofs for uncorrupted parties. By definition of NIZK argument systems, the simulator will fail with at most negligible probability. Therefore, ϵ = negl(λ) is negligible.

Lemma A.3 *For any PPT adversary $\mathcal{A}$, if IND-secure encryption schemes exist, then the advantage $|adv^{\mathcal{A}}_{H_3} - adv^{\mathcal{A}}_{H_2}| \leq \epsilon$ where ϵ is negligible probability.*

Proof. We replace the ballot ciphertexts $\{\overrightarrow{ct_i}\}_{i\in[\mathbb{V}]}$ with random ciphertexts $\{\widetilde{\overrightarrow{ct_i}}\}_{i\in[\mathbb{V}]}$. By definition, the encryption scheme is IND-secure. Therefore, the adversary $\mathcal{A}$ can distinguish Hybrid 3 from Hybrid 2 with negligible probability.

Lemma A.4 *For any PPT adversary $\mathcal{A}$, if bounded-collusion-secure encryption schemes exist and $|\mathbb{C}_c| \leq \delta_c$, then the advantage $|adv^{\mathcal{A}}_{H_4} - adv^{\mathcal{A}}_{H_3}| \leq \epsilon$ where ϵ is negligible probability.*

Proof. It's not hard to find that if malicious candidates can create the decryption keys of candidates in $\mathbb{C}_H$ in the real world with non-negligible probability, then in the ideal world, a simulator must exist that can solve the underlying hard problem of the BC-secure encryption scheme with non-negligible probability. To further illustrate the notion of bounded-collusion in our scheme, we take the instantiation in Appendix A.2 as a proof example.

Proof Sketch. Assume that a PPT adversary $\mathcal{A}$ who controls $\delta_c = |\mathbb{C}| - 1$ candidates break the security against *bounded-collusion* to reveal a voter's ciphertext with non-negligible advantage, there exists a PPT simulator $\mathcal{S}'$ can break the CDH problem with non-negligible advantage. Given the simulator an instance of computational Diffie-Hellman problem, i.e., (g, g^a, g^b). In the simulation world, the simulator $\mathcal{S}'$ does as follows:

1). Set the uncorrupted candidate $\mathbb{c}_{j^*}$'s public key $pk_{j^*} := g^a$.

2). Generate ballot ciphertexts $\{ct_{i^*,j}\}_{j\in[\mathbb{C}]}$, set one of the ciphertexts to $(ct^0_{i^*,j'} := g^b, ct^1_{i^*,j'} \leftarrow_R \mathbb{G})$, where j' denotes the identity of the candidate $\mathbb{c}_{j'}$. Thus, $\mathcal{S}'$ implicitly defines $v_{i^*,j'} = ct^1_{i^*,j'} \cdot \left(g^{b\cdot\sum_j^{[\mathbb{C}\backslash\mathbb{c}_{j^*}]} sk_j} \cdot g^{ab}\right)^{-1}$.

3). As in the OW-CPA security game[113], the simulator sends $\{ct_{i^*,j}\}_{j\in[\mathbb{C}]}$ to the adversary $\mathcal{A}$ and receives some ballot plaintexts $\breve{\vec{v}}_{i^*}$ containing $\breve{v}_{i^*,j'}$.

4). If $\mathcal{A}$'s guess is correct, then it holds $v_{i^*,j'} = \breve{v}_{i^*,j'} = ct^1_{i^*,j'}\left(g^{b\cdot\sum_j^{[\mathbb{C}\backslash\mathbb{c}_{j^*}]} sk_j} \cdot g^{ab}\right)^{-1}$ which implies $g^{ab} = ct^1_{i^*,j'} \cdot \left(\breve{v}_{i^*,j'} \cdot g^{b\cdot\sum_j^{[\mathbb{C}\backslash\mathbb{c}_{j^*}]} sk_j}\right)^{-1}$. Since $\mathcal{S}$ knows all $\{sk_j\}_{j\in[\mathbb{C}\backslash\mathbb{c}_{j^*}]}$(Thanks to the **Lemma A.1**), then g^{ab} can be computed efficiently.

5). If $\mathcal{A}$ wins, then also $\mathcal{S}'$ succeeds in solving CDH.

Lemma A.5 *For any PPT adversary $\mathcal{A}$, if BC-IND-secure encryption schemes exist and $|\mathbb{V}_c| \leq \delta_v$, then the advantage $|adv^{\mathcal{A}}_{H_4} - adv^{\mathcal{A}}_{H_3}| \leq \epsilon$ where ϵ is negligible probability.*

Proof. Assuming that $\delta_v = |\mathbb{V}| - 2$ voters with the same weight cast their ballots for different candidates. Due to the indistinguishability of the encryption scheme, if the adversary can distinguish, with non-negligible probability, which candidate the target secret ballot is cast for, then in the ideal world, a simulator must exist that can solve the underlying hard problem of the BC-IND-secure encryption scheme with non-negligible probability.

From all above, the views of these two worlds are indistinguishable.

Appendix B

B.1 Proof of Theorem 2.

Proof. We now show that under the assumption of computationally simulation-knowledge-sound zkSNARKs, secure SemiVoting construction, collision-resistant hash function in the Merkle tree, perfect binding and computational hiding commitment scheme and secure PRF scheme, then for every real-world adversary $\mathcal{A}$ against our scheme Π_{VV} there exists an ideal-world simulator $\mathcal{S}$ such that the view of the real-world and ideal-world adversaries are computationally indistinguishable. We will now describe the operation of the ideal-world simulator $\mathcal{S}$, which runs $\mathcal{A}$ internally and interacts with the ideal functionality $\mathcal{F}_{\mathrm{VV}}$.

We begin by sketching the operation of the ideal-world simulator $\mathcal{S}$. Due to the fact that VeriVoting is the extension of SemiVoting, only the corruption of dishonest voters is considered in this proof.

According to our scheme, the view of each dishonest voter $\mathbb{v}_i \in \mathbb{V}_c$ in the real world is as follows:

$$\mathrm{REAL}^{\Pi_{\mathrm{VV}}}_{\mathcal{A}_{\mathbb{C}}(z)}\left(\{\mathbb{v}_i\colon (addr_i, \vec{v}_i, w_i)\}_{i\in[\mathbb{V}_c]}, \{\mathbb{c}_j\colon \perp\}_{j\in[\mathbb{C}]}, \mathcal{SC}\colon \perp\right)$$
$$= \begin{Bmatrix} \{(addr_i, \vec{v}_i, w_i)\}_{i\in[\mathbb{V}_c]}, \{pk_j\}_{j\in[\mathbb{C}]}, pp_{sv}, crs, \{tx_{cm}, tx_{vote}\}_{i\in[\mathbb{V}]}, \\ \{tx_{dk}\}_{j\in[\mathbb{C}]}, \{t_j\}_{j\in[\mathbb{C}]} \end{Bmatrix}$$

To simulate the view of the candidate, the simulator $\mathcal{S}_{\mathbb{c}}$ does the following:

1) Run the $\mathrm{SV.Setup}(1^\lambda) \to \widetilde{pp_{sv}}$, and $(\widetilde{crs}_{PK}, \widetilde{crs}_{VK}, \tau) \leftarrow \Pi_{\mathrm{SNARK}}.\mathrm{Setup}_{\mathrm{sim}}(1^\lambda)$, and initialize two sets $\widetilde{\mathbb{CT}}$ and $\widetilde{\mathbb{DK}}$, and a list L_v for storing $(addr_i, w_i, \vec{v}_i)$ from $\mathbb{v}_i \in \mathbb{V}_c$.

2) Generate random public keys for uncorrupted candidates $\{\widetilde{pk}_j\}_{j\in[\mathbb{C}]}$ by random secret keys $\{\widetilde{sk}_j\}_{j\in[\mathbb{C}]}$.

On receiving $(addr_i, cm_i, w_i, s_i)$ from the voter in $\mathbb{V}_c$, append $(addr_i, w_i, \perp)$ into L_v.

3) If the voter is honest, randomly select $\widetilde{vsk}_i$ to generate $\widetilde{addr}_i$, then create $\widetilde{tx}_{cm} := (\widetilde{addr}_i, \widetilde{cm}_i, \widetilde{w}_i, \tilde{s}_i)$ as it was in the normal scheme.

4) On receiving $(\pi_i, \vec{ct}_i, sn_i)$ from the voter in $\mathbb{V}_c$, run the knowledge extractor on π_i to obtain $\vec{v}_i$ and $addr_i$, record them into L_v, then update $\widetilde{\mathbb{CT}}$.

5) Randomly select bogus ballot ciphertexts from ciphertext space $\mathcal{C}$, output $\widetilde{tx}_{vote} := \left\{\tilde{\pi}_i, \widetilde{\vec{ct}}_i, \widetilde{sn}_i\right\}_{i\in[\mathbb{V}\setminus\mathbb{V}_c]}$, where $\tilde{\pi}_i \leftarrow \Pi_{\mathrm{SNARK}}.\mathrm{Prove}_{\mathrm{sim}}(\widetilde{crs}_{PK}, \tau, (\widetilde{\vec{ct}}_i, \widetilde{sn}_i))$, $\widetilde{sn}_i$ is a random element from the codomain of PRF, then update $\widetilde{\mathbb{CT}}$.

6) Generate $\widetilde{tx}_{dk} := \left(\widetilde{\vec{dk}}_j, \tilde{\pi}_j\right)$ as in the proof of Theorem 1., and update $\widetilde{\mathbb{DK}}$.

7) The subsequent process is consistent with the scheme.

To prove indistinguishability of the real and ideal worlds from the perspective of the adversary $\mathcal{A}$, we will go through a sequence of hybrid games. In both views, the tally result part is identical.

H_0: This is the real world.

H_1: In this hybrid, we replace ciphertexts $\{\pi_i, \vec{ct}_i, sn_i\}_{i\in[\mathbb{V}\setminus\mathbb{V}_c]}$ with $\left\{\tilde{\pi}_i, \widetilde{\vec{ct}}_i, \widetilde{sn}_i\right\}_{i\in[\mathbb{V}\setminus\mathbb{V}_c]}$. By Lemma B.1, if the zkSNARK argument system is computational zero-knowledge, the encryption scheme is BC-IND-secure and PRF is a pseudorandom function, then $\mathrm{H}_0 \overset{c}{\approx} \mathrm{H}_1$.

H_2: In this hybrid, we run the knowledge extractor when encountering the output of any corrupted voters $(\pi_i, \vec{ct}_i, sn_i)$, and abort if the knowledge extractor fails. By Lemma B.2 we show that if the proof extractor fails with negligible probability, then $\mathrm{H}_1 \overset{c}{\approx} \mathrm{H}_2$.

H_3: In this hybrid, we replace the uncorrupted votes' tx_{cm} with $\widetilde{tx}_{cm}$ by running the normal scheme. Hybrid 3 is identical to Hybrid 2, then $\mathrm{H}_2 = \mathrm{H}_3$.

H_4: In this hybrid, we replace all decryption keys and proofs $\{\vec{dk}_j, \pi_j\}_{j\in[\mathbb{C}_H]}$ with simulated proofs $\left\{\widetilde{\vec{dk}}_j, \tilde{\pi}_j\right\}_{j\in[\mathbb{C}_H]}$. By Lemma B.3, if the NIZK argument system is computational zero-knowledge, then $\mathrm{H}_3 \overset{c}{\approx} \mathrm{H}_4$.

Note that Hybrid 4 is identical to the simulation. By summation over the previous hybrids we show that $\mathrm{H}_0 \overset{c}{\approx} \mathrm{H}_4$. We conclude our proof sketch by presenting the supporting lemmas.

Lemma B.1 *For any PPT adversaries $\mathcal{A}$, if computational zero-knowledge SNARK systems, BC-IND-secure encryption schemes and secure PRFs exist, then advantage $\left|adv^{\mathcal{A}}_{H_1} - adv^{\mathcal{A}}_{H_0}\right| \le \epsilon$ where ϵ is negligible probability.*

Proof. The simulator operates in the same manner, but we now simulate proofs for uncorrupted parties. By definition of NIZK argument systems, the simulator will fail with at most negligible probability. The ciphertexts $\vec{ct}_i$ and serial number sn_i are randomly chosen, then the probability that the adversary can distinguish them from the real world is also negligible.

Lemma B.2 *For any PPT adversary $\mathcal{A}$, if simulation sound extractable zkSNARK systems exist, then advantage $\left|adv^{\mathcal{A}}_{H_2} - adv^{\mathcal{A}}_{H_1}\right| \le \epsilon$ where ϵ is the extraction failure probability.*

Proof. The simulator operates in the same manner, but we now extract when given $(\pi_i, \vec{ct}_i, sn_i)$ of corrupted voters who want to generate illegal encrypted ballots $\vec{ct}_i$, i.e., invalid statements, to change the tally result. By definition, the extractor will fail with at most negligible probability ϵ because it deals solely with zkSNARKs, which have efficient extractors. Therefore, our proofs have knowledge extractors that succeed with probability $1 - 1/\mathrm{negl}(\lambda)$.

Lemma B.3 *For any PPT adversaries $\mathcal{A}$, if computational zero-knowledge NIZK argument systems Π_{NIZK} exist, then advantage $\left|adv^{\mathcal{A}}_{H_4} - adv^{\mathcal{A}}_{H_3}\right| \le \epsilon$ where ϵ is the simulation failure probability.*

Proof. The simulator operates in the same manner, but we now simulate proofs for uncorrupted candidates. By definition of the NIZK argument Π_{NIZK} systems, the simulator will fail with at most negligible probability. Therefore, $\epsilon = 1/\mathrm{negl}(\lambda)$ is negligible.

From all above, the views of these two worlds are indistinguishable.

B.2 Discussion (Cont.).

In Section 6, we mainly discussed decentralization, verifiability, and privacy-preserving. In this section, we continue to discuss the other properties that the scheme has.

Eligibility. It means that only permitted voters can submit ballots. In the registration phase of VeriVoting, the voter should present his/her valid credential to $\mathcal{EA}$ to identify himself/herself as a valid voter. After confirming the legal identity of the voter in online or offline ways, $\mathcal{EA}$ issues a token to the voter by submitting a transaction to the blockchain. Then, the token denotes the eligibility of the ballot the voter will cast. In VerVoting, the voter can prove the possession of his/her token in zero-knowledge, which makes the identity and the token unlinkable.

Transparency. It means that the outputs of voters and candidates are transparent and verifiable for the public. With the blockchain and smart contract's transparency, external or internal observers can monitor the whole voting process in VeriVoting.

Fairness. It means that the tally results are not counted in real-time. In other words, if some candidate knows the tally results before the tallying phase, it will take some countermeasures to undermine the election's fairness, such as vote buying, coercion, etc. In VerVoting, we distribute the decryption power to all competitors. Because of the competition, no one wants others to know the results in advance. Moreover, VerVoting adopts (δ_c, δ_v)-secure SemiVoting scheme, which guarantees ballot privacy so no one can learn the ballot content.

Receipt-Freeness. It means that a voter cannot prove to anyone how she voted. In our scheme, we ignore the case where voters sell their voting eligibility to others, that is, to authorize the wallet address to others, since this property cannot be satisfied by any scheme in that case. Through the interaction with the contract, a voter only knows that his ballot has been verified and aggregated with other ballots. And with bidirectional unlinkability, no one can determine which ballot is the voter's ballot.

Duplicate Voting Detection. It means that each eligible voter is allowed to vote only once. As discussed in the commitment and voting phase in Section 3.4, as long as the serial number sn can not be forged, there is no possibility of duplicate voting. The security of PRFs guarantees this property.

Dispute-Freeness. It means that the tally result is publicly verifiable. Transparency implies that anyone can compute the tally result by verified decryption keys and ballot ciphertexts. Therefore, the property holds in VeriVoting.