Chinese Journal of Electronics
vol. 33, no. 4, pp. 1063–1076, July 2024
https://doi.org/10.23919/cje.2023.00.233

RESEARCH ARTICLE

A Distributed Self-Tallying Electronic Voting System Using the Smart Contract

Jingyu YAO[1,2], Bo YANG[1], Tao WANG[1,2,3], and Wenzheng ZHANG[3]

1. *School of Computer Science, Shaanxi Normal University, Xi'an 710119, China*
2. *State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an 710119, China*
3. *Science and Technology on Communication Security Laboratory, Chengdu 610041, China*

Corresponding author: Jingyu YAO, Email: jingyuyao@snnu.edu.cn
Manuscript Received July 2, 2023; Accepted December 8, 2023

Abstract — For electronic voting (e-voting) with a trusted authority, the ballots may be discarded or tampered, so it is attractive to eliminate the dependence on the trusted party. An e-voting protocol, where the final voting result can be calculated by any entity, is known as self-tallying e-voting protocol. To the best of our knowledge, addressing both abortive issue and adaptive issue simultaneously is still an open problem in self-tallying e-voting protocols. Combining Ethereum blockchain with cryptographic technologies, we present a decentralized self-tallying e-voting protocol. We solve the above problem efficiently by utilizing optimized Group Encryption Scheme and standard Exponential ElGamal Cryptosystem. We use zero-knowledge proof and homomorphic encryption to protect votes' secrecy and achieve self-tallying. All ballots can be verified by anyone and the final voting result can be calculated by any entity. In addition, using the paradigm of score voting and "1-out-of-k" proof, our e-voting system is suitable for a wide range of application scenarios. Experiments show that our protocol is more competitive and more suitable for large-scale voting.
Keywords — Self-tallying, Electronic voting system, Distributed voting, Smart contract.
Citation — Jingyu YAO, Bo YANG, Tao WANG, *et al.*, "A Distributed Self-Tallying Electronic Voting System Using the Smart Contract," *Chinese Journal of Electronics*, vol. 33, no. 4, pp. 1063–1076, 2024. doi: 10.23919/cje.2023.00.233.

I. Introduction

Compared to traditional election methods, electronic voting (e-voting) has great advantages due to its freedom from space constraints. There are many application scenarios for e-voting, such as board elections, crowdsourcing [1], crowdsensing [2], feature selection [3]–[5], extreme learning machine [6] and result filtering [7], [8]. It can be seen that e-voting is very important. Many scholars have made great contributions to the development of e-voting [9]–[12].

A blockchain is a public, append-only, immutable ledger, which plays an important role in e-voting protocols [13], [14]. Compared to election protocols [15]–[17] with a trusted authority, e-voting protocols [10]–[13], [18] without trusted parties have more advantages in protecting privacy and ensuring security, where blockchain and Group Encryption Scheme play important roles.

In decentralized e-voting protocols, self-tallying is an important property, which was introduced by Kiayias *et al.* in 2002 [9]. It means that the voting result can be calculated by any entity. Every step in self-tallying e-voting protocols can be verified to ensure correctness and dispute freeness. To the best of our knowledge, the abortive issue or adaptive issue are the problems faced by most self-tallying e-voting protocols [9], [12], [13], [16], [18]–[22]. In addition, due to high computational overhead, some e-voting protocols [12], [19] are not suitable for large-scale voting. Using Intel Software Guard Extensions (SGX), Yang *et al.* [10] designed an e-voting protocol without the abortive issue and the adaptive issue. But the security of their solution depends entirely on SGX and the reliance on SGX limits the application scenarios of their scheme.

In this paper, we propose a self-tallying e-voting protocol using a smart contract deployed on Ethereum, which can be replaced by any blockchain that supports

Associate Editor: Prof. Debiao HE, Wuhan University.

smart contracts. We use the paradigm of score voting introduced in [12] to make our scheme suitable for more scenarios. In addition, we have optimized the Group Encryption Scheme [11], [18] through sorting to make it more suitable for our system. By combining optimized Group Encryption Scheme with standard Exponential ElGamal Cryptosystem, the abortive issue and the adaptive issue are effectively solved simultaneously. Our scheme can tolerate interruptions by any number of voters and the last voter cannot get the voting result in advance. Moreover, our solution does not require any trusted party.

1. Contributions

The contributions of this paper are summarized as follows:

1) We present a secure, verifiable, and self-tallying e-voting protocol where any entity can verify the ballots and calculate the voting result. In addition, our system does not require a trusted administrator.

2) The abortive issue and the adaptive issue are solved efficiently by utilizing optimized Group Encryption Scheme and standard Exponential ElGamal Cryptosystem. If there are some voters who do not vote, they will be removed and the rest active voters can restore the voting process efficiently and complete voting. More details about this case can be found in Section V. For the adaptive issue, the last voter cannot get the final voting result in advance without collusion with the administrator. The voting result can only be calculated after the administrator's secret key is published.

3) Experiment shows that in our voting system, computation overhead of each voter is lower and the proof size is smaller compared to the works closely related to ours. And of course, we have a faster verification process. So we believe our scheme is suitable for large-scale voting.

2. Structure of this paper

The rest of this paper is structured as follows. Section II presents the research closed to ours. Section II describes the cryptographic techniques used in our paper. Section IV shows the system and threat model of our scheme. Section V presents the concrete construction of our system. Section VI shows the security analysis of our protocol. Section VII presents the efficiency analysis of our scheme and the comparison with two closed protocols. Section VIII shows a summary of our protocol and the prospects of our future research.

II. Related Work

In this section, we have compared our work with the research closely related to ours. The origin of the self-tallying voting system can date back to the work of Kiayias *et al.* [9], who proposed an election paradigm and realized the first self-tallying voting protocol where both strong ballot privacy and fault tolerance were captured. For a candidate in [9], each voter needs to select n random values $r_{i,j} \in \mathbb{Z}_q$ (n is the number of voters and $\sum_{j=1}^{n} r_{i,j} = 0$). Because the computational overhead of per voter for each candidate has size proportional to the number of voters, this election protocol is not suitable for large-scale election. References [11]–[13], [19], and [21] realize self-tallying voting protocols using similar cryptographic techniques In these schemes, the computation overhead of voters is too large to support large-scale elections and the overhead of obtaining public keys from the blockchain is expensive, each voter needs to compute $\prod_{j=1}^{i-1} y_j / \prod_{j=i+1}^{n_v} y_j$ where n_v is the number of voters and y_j is the public key of voter v_j. Compared to above works, Zeng *et al.* [18] reduced user's computational overhead while maintaining self-tallying, each voter needs to calculate $(g^{(y_i)x_i}/g^{(x_j)y_j})g^{v_m}$ where $(x_i,\ y_j)$ are voter's secret keys, $(g^{y_i},\ g^{x_j})$ are public keys published by other voters, and v_m is the number of the candidates. It is obvious that the computational overhead of voters is very low. By optimizing the idea in [18] and combining it with other cryptographic techniques, our system is more suitable for large-scale voting compared to the above works and the following problems are solved by us.

Some of the problems are caused by the trusted setting. For genetic model of e-voting protocol in [23], the clear-text ballots are encrypted by the polling office using a blinding share and the sum of all shares is equal to zero. Then the polling office and the voters sign the blinded ballot and anyone can decrypt it and get the blinded ballot. By adding all the blinded ballots, anyone can get the final result of voting, which meets the characteristics of self-tallying. But the polling office can get all the clear-text ballots in advance, which may undermine the fairness of the scheme. The protocol in [17] regards money as a voting tool and the transactions between voters and candidates as ballots without modifying the construction of Zcash protocol. Inheriting the characteristics of Zcash, this election protocol realizes transparency and anonymity. But the trusted organizer may give voters different amounts of currency to undermine the fairness of voting. Yu *et al.* [14] presented an e-voting protocol where the anonymity is realized by using short linkable ring signatures (SLRS). It realizes the independence of the platform and can be used for large-scale elections. However, the administrator in voting protocol can obtain the candidates' information as it is encrypted by the administrator's Paillier public key, which may break the fairness of their e-voting protocol. Chaieb *et al.* [15] designed a voting protocol where a trusted registration server and an election administrator are needed. In [15], each voter needs to be authorized by the trusted party and the authorization is the information encrypted by the trusted party using the private key, so the privacy of votes may be compromised. With Shamir's secret sharing proposed in [24], Huang *et al.* [16] proposed a self-tallying scheme where the privacy of voters is protected under certain conditions. If the election authority conspires with any candidate, the voting content of that candidate can be restored in advance. In [25], a trusted counting node is responsible for decrypting the voter's encrypted data and counting the votes. Each voter's voting information is encrypted using the counting node's public key, so the security of the protocol depends on the security of the counting node. As the counting node can get the voting information in advance, it is difficult to guarantee fairness and verifiability of the scheme. Compared above protocols [14]–[17], [25], the administrator in our scheme is trustless, which means the administrator cannot affect the security of our protocol.

The abortive issue and the adaptive issue are also important problems faced by many voting systems. The protocols in [11], [12], [19], [21], [22], and [26] cannot resist abortive issue for more than one voter and clearly the adaptive issue remains to be solved. Although the adaptive issue in [13] is solved, the abortive issue is not fundamentally resolved by requiring absentee voters to cast a zero-point ballot to a candidate set in advance. With similar technology in [18], reference [20] further solves the collusion of neighboring voters and the absence of some voters, but it cannot solve the absence of adjacent voters. Javani *et al.* [26] proposed an e-voting system called boardroom voting with oblivious transfer (BVOT) where the main novel feature is its use of oblivious transfer (OT) to provide perfect ballot secrecy and ensure correct vote casting. Doing so avoids the need for voters to carry out or verify complex zero-knowledge proofs. It reduces the number of checks and verifications by hiding the information necessary for cheating from the adversary. Compared to our work, this scheme has a significant abortive issue, as it cannot calculate the final voting result when any voter does not vote. In addition, the voting process of this system is more complex. After the voting phase, all voters must publish a decryption share which will be used to calculate the final result. In our system, anyone can directly calculate the voting result through voting information with the administrator's secret key. Moreover, as described in the BVOT article, its design is suitable for small-scale voting scenarios, such as board elections, while our design can be suitable for large-scale voting scenarios. Compared to above works, Both the abortive issue and the adaptive issue are solved efficiently in our protocol. We can solve the abortive issue of any voter with low computational overhead.

Xue *et al.* designed an e-voting system called ACB-Vote (ACB, anonymously convertible ballots) [27]. Although the ACB-Vote does not use self-tallying technology, it achieves anonymity, fairness, and universal verifiability using the cryptographic tools such as BBS+ signature, collision-resistant hash algorithm, and convertibly linkable signatures. ACB-Vote has no abortive issue and adaptive issue. However, ACB-Vote uses two non-collude converters who are responsible for ballot conversion to prevent duplicate voting while maintaining user anonymity. This is inevitably required due to the anonymity of voters and the exposure of the ballots' contents in tallying phase. This undoubtedly increases system security risk. In our system, voting content is not exposed even during the tallying stage, so, we do not need the converters, which makes our system more elegant and more secure. Chentouf *et al.* [28] proposed a secure e-voting system based on blockchain. By utilizing the transparency and tamper resistance of blockchain, this voting system ensures the security of voting to a certain extent. However, the voting content is stored in clear text on the blockchain, which is not conducive to protecting the privacy of voters. In addition, This can cause the adaptive issue. In our system, the ballots' contents of voters will not be exposed at any stage of voting, the privacy of voters is well protected, and our solution does not have the adaptive issue. The constructions of e-voting systems in the works [29]–[31] are very similar. They have detailed and strict regulations for the management of users' identity. The voting information is recorded on the blockchain. This can prevent tampering with voting information, but cannot guarantee the privacy of voters. Due to the independence of each voter, interrupted voters will not affect the final vote counting stage. With the fact that each voter can see the voting information of other voters during voting, this scheme has the adaptive issue.

With Intel SGX, Yang *et al.* [10] presented a distributed SGX networked system (DSGXNS) to realize e-voting. The abortive issue is naturally resolved with SGX. The voters in [10] do not need to calculate their own ballots, but to send the scores of candidates and corresponding random numbers to SGX in an encrypted manner, then SGX will calculate the ballots of each voter and publish them on the bulletin board. The security of the protocol depends on the security of SGX and the inevitable requirement for SGX will limit the application scenarios of the voting protocol. In Table 1, we compare our work with closed e-voting protocols [9]–[19], [21], [22], [25] from thirteen aspects, including efficiency, properties, and security requirements. For computational overhead in voting phase, although the cost of each voter in our protocol scales linearly in the number of candidates, the cost is low due to the limited number of candidates in reality. In addition, there are no trusted authorities in our scheme. Moreover, the abortive issue and the adaptive issue are effectively solved simultaneously. Although Yang *et al.* [10] also solved these two problems, their scheme relies heavily on SGX.

III. Preliminary

1. Exponential ElGamal Cryptosystem

SysGen The system parameter generation algorithm takes as input a security parameter λ. It chooses a cyclic group $(\mathbb{G}, p, g)$ and returns the system parameters $\mathrm{SP} = (\mathbb{G}, p, g)$.

KenGen The key generation algorithm takes as input the system parameters SP. It ramdomly chooses

Table 1 Comparison of electronic voting protocols

Scheme	Properties												
	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13
[12]	C1	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Multiple	Score voting	No
[18]	C2	Yes	Yes	No	C7	No	Yes	Yes	Yes	Yes	Multiple	1 from n_c	Yes
[13]	C3	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Multiple	1 from n_c	No
[15]	C4	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Multiple	C7	Yes
[19]	C3	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Single	Yes or No	No
[11]	C1	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Multiple	Score voting	No
[9]	C2	Yes	Yes	Yes	Yes	C8	Yes	Yes	Yes	Yes	Single	Yes or No	No
[17]	C5	No	No	No	C7	Yes	Yes	Yes	No	Yes	Multiple	m from n_c	Yes
[14]	C5	Yes	No	Yes	C7	No	No	Yes	Yes	Yes	Multiple	1 from n_c	Yes
[25]	C5	No	No	Yes	C7	Yes	No	No	No	C7	Multiple	m from n_c	Yes
[21]	C3	Yes	Yes	Yes	C7	No	Yes	Yes	Yes	Yes	Single	Yes or No	No
[10]	C5	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Multiple	Score voting	Yes
[16]	C5	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Multiple	Score voting	Yes
[22]	C3	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Multiple	1 from n_c	Yes
This work	C6	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Multiple	Score voting	Yes

Note: A1: Voting phase computation; A2: No trusted party; A3: Self-tallying; A4: Platform independent; A5: No abortive issue; A6: No adaptive issue; A7: Fairness; A8: Individual verifiability; A9: Universal verifiability; A10: Multiple-voting detection; A11: Candidates; A12: Voting type; A13: Large-scale election; C1: $O\left(n_v^2 n_c\right) Ex$; C2: $O\left(n_v n_c\right) Ex$; C3: $O\left(n_v^2\right) Ex$; C4: 2 En + 1 Sign; C5: Constant; C6: $O\left(n_c\right)$; C7: Not specified.

$\alpha \in \mathbb{Z}_p$, computes $g_1 = g^\alpha$, and returns a public/secret key pair (pk, sk) as follows:

$$\mathrm{pk} = g_1, \quad \mathrm{sk} = \alpha$$

Encrypt The encryption algorithm takes as input a message $m \in \mathbb{G}$, the public key pk, and the system parameters SP. It chooses a random number $r \in \mathbb{Z}_p$ and returns the ciphertext CT as

$$\mathrm{CT} = E(m) = (C_1,\ C_2) = (g^r,\ g_1^r \cdot g^m)$$

Decrypt The decryption algorithm takes as input a ciphertext CT, the secret key sk, and the system parameters SP. Let $\mathrm{CT} = (C_1, C_2)$. It decrypts the message by computing

$$C_2 \cdot C_1^{-\alpha} = g_1^r g^m \cdot (g^r)^{-\alpha} = g^m$$

Homomorphism ElGamal encryption has an inherited homomorphic property, which allows multiplication and exponentiation to be performed on a set of ciphertexts without decrypting them, such as

$$\begin{aligned} E(m_1) \cdot E(m_2) &= (g^{r_1}, g_1^{r_1} \cdot g^{m_1}) \times (g^{r_2}, g_1^{r_2} \cdot g^{m_2}) \\ &= (g^{r_1+r_2}, g_1^{r_1+r_2} \cdot g^{m_1+m_2}) = E(m_1 + m_2) \end{aligned}$$

2. Zero knowledge proof

Based on the non-interactive proof of knowledge of a secret in [32] and the 1-out-of-k zero knowledge proof in [12], We construct a zero knowledge proof protocol for our scheme. Each voter must prove that he/she knows the secret values of the ballots, we use PoK$\{x : g^x\}$ to denote a non-interactive zero knowledge proof of the secret x. In addition, The score in each ballot should be within a specific range, we use $\mathrm{SC} = \{p_1, p_2, \ldots, p_n\}$ to denote a public set of n values and OoKPoK$\{\exists p_i \in \mathrm{SC} : c = g^{p_i} \mathrm{pk}^r\}$denote the 1-out-of-$k$ zero knowledge proof where the ciphertext c is the result of encrypting an element p_i in the set S.

3. Group Encryption Scheme

Figure 1 presents one of key ideas of our e-voting protocol. Each voter i $(i \in [1, n_v])$ has to select two secret keys (x_i, y_i) and submits the corresponding public keys (g^{x_i}, g^{y_i}). Then the smart contract randomly sorts the public keys and the generates a list of voters. Each voter i needs to calculate $Y_i = g^{y_i x_{i+1} - x_i y_{i-1}}$ secretly. Note that the first voter in the row calculates $Y_1 = g^{y_1 x_2 - x_1 y_n}$ and the last voter in the row calculates $Y_n = g^{y_n x_1 - x_n y_{n-1}}$. We have $\prod_{i=1}^n Y_i = 1$. We combine this technology with Exponential ElGamal Cryptosystem mentioned in Section III.1 to construct our e-voting system.

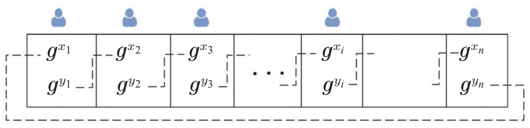


Figure 1 Public keys graph.

IV. System and Threat Model

In this section, We first present our system model, and then introduce the threat model and security goals of our scheme.

1. System model

In Figure 2, we present the system model of our e-voting protocol. The main roles in the system model are as follows.

Ethereum Ethereum is an open source public blockchain platform with smart contract functionality, providing decentralized Ethereum virtual machines to handle peer-to-peer contracts. In our protocol, it is used as the executor of the rules and as a public bulletin board. Note that Ethereum can be replaced by any blockchain that supports smart contracts.

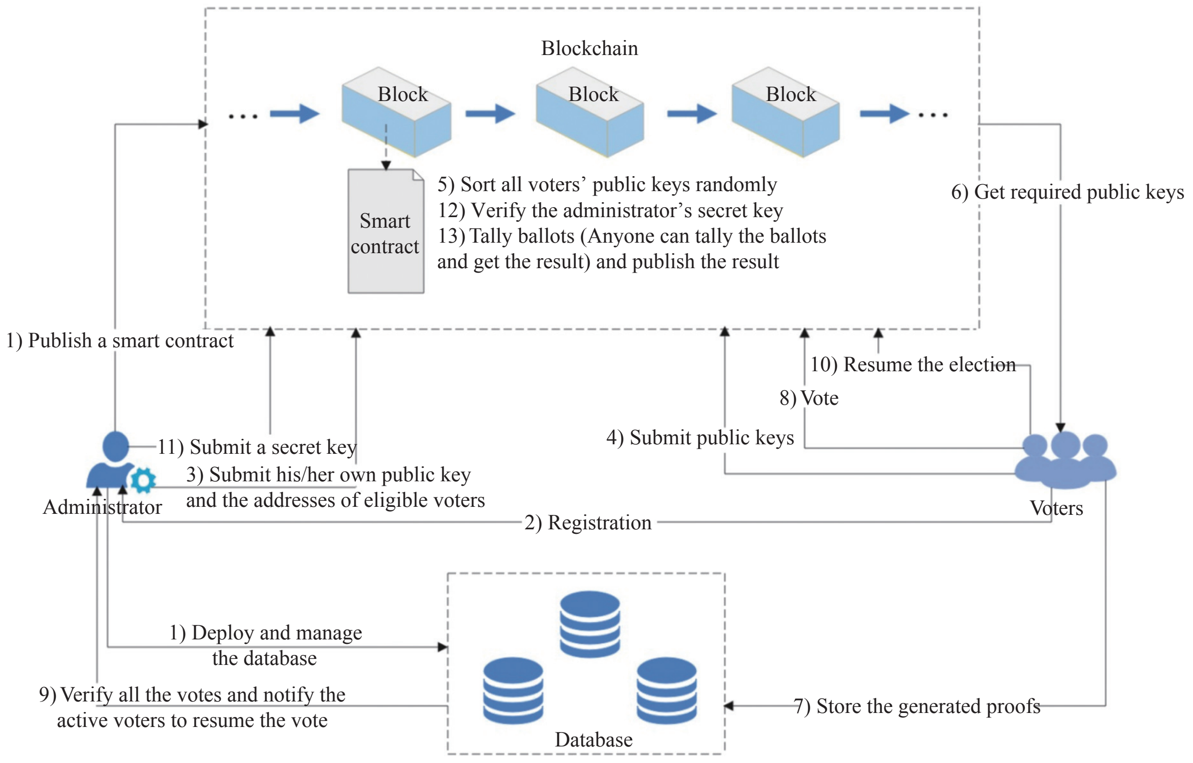

Blockchain
Block
Block
Block
Smart contract
5) Sort all voters' public keys randomly
12) Verify the administrator's secret key
13) Tally ballots (Anyone can tally the ballots and get the result) and publish the result
6) Get required public keys
1) Publish a smart contract
10) Resume the election
8) Vote
4) Submit public keys
11) Submit a secret key
3) Submit his/her own public key and the addresses of eligible voters
Administrator
Voters
2) Registration
1) Deploy and manage the database
9) Verify all the votes and notify the active voters to resume the vote
7) Store the generated proofs
Database

Figure 2 System model.

Smart contract Smart contract is an open, transparent, and unchangeable protocol deployed on the blockchain. One of its functions is to serve as a public bulletin board. In our protocol, the smart contract published by the administrator maintains five lists, namely, a list of active voters who did not vote (NVAVL), a list of active voters who have voted (HVAVL), a negative voters list (NVL), a voters' list for sorting (SVL), and a list of voters who can help restore voting process (RVPVL). In addition, the smart contract in our e-voting protocol also includes a function that can only be applied by the administrator to submit his/her own public key or secret key, a function that can only be applied by the administrator to submit the addresses of eligible voters, a function used by the voters to submit their public keys, a function used by the smart contract itself for sorting voters in the SVL, a voting function that can only be used by voters in the NVAVL, a function to verify the administrator's secret key, and a function to tally ballots. Only voters in the NVAVL can vote and can only vote once. Once a voter in NVAVL votes, his/her address will be added to the HVAVL and deleted from the NVAVL. During the phase of voting, the voters who do not vote or submit incorrect ballots will be added to the NVL and removed from the NVAVL, then the adjacent voters will be added to the RVPVL and they have to resume the voting process. After all active voters have voted and all the ballots have been verified, the smart contract can help calculate the voting result.

Administrator The administrator is responsible for deploying the smart contract with initial parameters, adding addresses of eligible voters to the smart contract, submitting his/her own public key and secret key at the specified time, verifying all the ballots submitted by the voters, opening voting and closing voting. The initial parameters include the administrator's address, the set of scores, the number of voters, each voter's identification information, the number of candidates, and the cyclic group $(\mathbb{G}, p, g)$.

Voters Each voter needs to generate two private keys and two corresponding public keys. The public keys need to be submitted to the smart contract before voting. The voters can call the voting function in the smart contract to vote and the hash of the proof about the ballots should also be stored on the blockchain at the same time. The proof portion of the ballots needs to be stored in the database because storing it on the blockchain is too expensive. In addition, when there are voters in the NVAVL, voters in the RVPVL must help resume the voting process.

Database Considering the problem that storing the proof in the blockchain is too expensive, the proof can be stored in the database to reduce the cost of the voters. Each voter only needs to store the hash value of the proof on the blockchain. Since anyone can verify that the data in the database is correct through the hash value on the blockchain, the immutable nature of the blockchain and the security of the one-way hash function guarantee the security of the proof in the database. As the security of the database does not depend on the one who manages it, we put administrators in charge of deploying and managing the database.

The notations in Table 2 are used in the rest of the paper.

Table 2 Notations

Notations	Explanation
n	Number of voters
m	Number of candidates
i	$i \in [1, n]$
j	$j \in [1, m]$
r_i^j	A random number selected by the i-th voter for the j-th candidate
x_i	Secret key of the i-th voter
y_i	Secret key of the i-th voter
g^{x_i}	Public key of the i-th voter
g^{y_i}	Public key of the i-th voter
pk	A public key of the administrator
sk	A secret key of the administrator
S	The max value of scores
p_i^j	A score that is assigned by the i-th voter to the j-th candidate; $p_i^j \in S$
p^j	Total received scores of the j-th candidate
c_{1i}^j	The first part of the ciphertext generated by the i-th voter to the j-th candidate
c_{2i}^j	The second part of the ciphertext generated by the i-th voter to the j-th candidate

2. Threat model

The administrator is an untrustworthy party who may want to get the contents of ballots. Besides, the administrator may manipulate the voting result, including accepting illegal voters or requiring corrupt voters to vote illegally. Moreover, the administrator may modify the data in the database to make legitimate voting impossible to pass verification.

The voters may attempt to get the voting contents of others during voting or try to forge ballots. Besides, the voters may attempt to cast more than one ballot in an election or make his scores exceed the regulated range. Moreover, the voters may do not vote after registration.

We require that the administrator cannot collude with two voters separated by one voter in the sorted voters' list. In addition, the administrator cannot collude with the last voter in the voters' list sorted by the smart contract.

3. Security goals

The security goals of our protocol are described below.

1) Ballots secrecy: Except for the owners of the ballots, no entity can obtain the voting contents of the voters.

2) Fairness: Fairness means that no one can obtain intermediate information before tallying ballots.

3) No abortive issue: The voting can still be done if some voters do not submit their votes.

4) No adaptive issue: The last voter cannot obtain any information about the voting result in advance.

5) Eligibility: Only eligible voters can vote.

6) Integrity: No one can change any submitted ballots or use them to create duplicate submissions.

7) Multiple-voting detection: Each voter can only vote once. Extra ballots will be rejected.

8) Dispute freeness: Invalid ballots can be detected by any entity.

9) Verifiability: It requires that each voter is able to verify whether the ballots are correctly published and tallied. Any entity can verify the final voting result.

V. Concrete Construction

1. Initialization and voter registration

The first step is to deploy the smart contract by the administrator. The administrator must assign values to the initialization variables, which include the administrator's address, the set of scores S, the number of voters n, each voter's identification information, the number of candidates m, and the cyclic group $(\mathbb{G}, p, g)$. All initial parameters and functions in the smart contract are transparent, and anyone can check the correctness of them.

Each voter must generate two ElGamal key pairs (x_i, g^{x_i}) and (y_i, g^{y_i}) where $x_i \in \mathbb{Z}_p$ and $y_i \in \mathbb{Z}_p$. After proving the identity is eligible, a voter's address will be submitted to the NVAVL and SVL in smart contract. Then, each voter submits his/her public keys to the smart contract. After receiving all the public keys, the smart contract sorts them randomly and the result is shown in Figure 1. For convenience, we denote the sorting result as "circle". Finally, the administrator can open the voting phase.

2. Voting

For voting, there are three basic phases: pre-voting, vote cast, and proof generation. Let's take the i-th voter as an example.

1) Pre-voting: Voter i needs to get the number of candidates m, the public keys $(g^{x_{i+1}}, g^{y_{i-1}})$, and the administrator's public key pk from the smart contract. Then, voter i has to select m random numbers r_i^j $(j \in [1, m])$.

2) Vote: For each candidate $j (j \in [1, n_c])$, voter i can assign an integer score p_i^j where $p_i^j \in [0, S]$. For each candidate j, voter i calculates

$$E(p_i^j, r_i^j, \mathrm{pk}) = (g^{r_i^j}, g^{p_i^j} \cdot \mathrm{pk}^{r_i^j})$$

using Exponential ElGamal Encryption mentioned in Section III.1. Then the $g^{r_i^j}$ is encrypted again with voter i's secret keys (x_i, y_i) and the public keys $(g^{x_{i+1}}, g^{y_{i-1}})$. The result is

$$E(g^{r_i^j}, x_i, y_i, g^{x_{i+1}}, g^{y_{i-1}}) = g^{y_i x_{i+1} - x_i y_{i-1}} \cdot g^{r_i^j}$$

We can summarize the encryption algorithm as

$$\begin{aligned} &E(p_i^j, r_i^j, \mathrm{pk}, x_i, y_i, g^{x_{i+1}}, g^{y_{i-1}}) \\ &= (g^{y_i x_{i+1} - x_i y_{i-1}} \cdot g^{r_i^j}, g^{p_i^j} \cdot \mathrm{pk}^{r_i^j}) = (c_{1i}^j, c_{2i}^j) \end{aligned}$$

where p_i^j is the score given to the candidate j by the voter i, r_i^j is the random number used by the voter i for the candidate j, pk is the public key of the administrator, (x_i, y_i) are the secret keys of the voter i, and $(g^{x_{i+1}}, g^{y_{i-1}})$ are the public keys of adjacent voters in the circle. The final ballots of the voter i can be presented as

$$\begin{gathered} E(p_i^1, r_i^1, \mathrm{pk}, x_i, y_i, g^{x_{i+1}}, g^{y_{i-1}}) = (c_{1i}^1, c_{2i}^1) \\ \vdots \\ E(p_i^m, r_i^m, \mathrm{pk}, x_i, y_i, g^{x_{i+1}}, g^{y_{i-1}}) = (c_{1i}^m, c_{2i}^m) \end{gathered}$$

The summarised processing procedure of the voting stage is shown in Algorithm 1.

Algorithm 1 Vote

```
Require:
    Input: public keys $(g^{x_{i+1}}, g^{y_{i-1}})$, all random numbers $\{r_i^j\}_{j=1}^m$, secret keys $(x_i, y_i)$, the administrator's public key pk and scores corresponding to all candidates $\{p_i^j\}_{j=1}^m$;
    Output: $\{c_{1i}^j, c_{2i}^j\}_{j=1}^m$;
1: Compute $g^{y_i x_{i+1} - x_i y_{i-1}}$;
2: for $j \leftarrow 1$ to $m$ do
3:     $E(p_i^j, r_i^j, \mathrm{pk}, x_i, y_i, g^{x_{i+1}}, g^{y_{i-1}}) = (g^{y_i x_{i+1} - x_i y_{i-1}} \cdot g^{r_i^j}, g^{p_i^j} \cdot \mathrm{pk}^{r_i^j})$;
4: end for
5: return $\{c_{1i}^j, c_{2i}^j\}_{j=1}^m$.
```

3) Proof generation: In order to allow anyone to verify the eligibility of ballots without decrypting the ciphertext and revealing the content, each voter is required to generate proof for ballots.

a) PoK$\{-x_i : g^{-x_i}\} \wedge$ PoK$\{y_i : g^{y_i}\}$: The voter i is required to prove that the ballots are computed correctly using the private keys (x_i, y_i). The process is as follows:

- Select three random numbers $k_1, k_2, k_3 \in \mathbb{Z}_q$, compute $T_1 = g^{k_1}$, $T_2 = g^{k_2}$, $T_3 = (g^{x_{i+1}})^{k_1} \cdot (g^{y_{i-1}})^{k_2} \cdot g^{k_3}$, $c = \mathrm{Hash}(T_1 \parallel T_2 \parallel T_3)$, $Z_1 = y_i c + k_1$, $Z_2 = -x_i c + k_2$, and $Z_3 = r_i^j c + k_3$. The proof is PoK$\{-x_i : g^{-x_i}\} \wedge$ PoK$\{y_i : g^{y_i}\} = \{T_1, T_2, T_3, Z_1, Z_2, Z_3\}$.
- As the voter i needs to generate m proofs as above corresponding to m candidates and there are many similarities between proofs, voter i can combine all the proofs as follows: For each vote, voter i calculates $Z_3^j = r_i^j c + k_3$ where $j \in [1, m]$. $\{T_1, T_2, T_3, Z_1, Z_2\}$ is universal. The final proof is $\pi_1 = \{T_1, T_2, T_3, Z_1, Z_2, \{Z_3^j\}_{j=1}^m\}$.

b) OoKPoK$\{\exists p_i^j \in [0, S] : c_{2i}^j = g^{p_i^j} \cdot \mathrm{pk}^{r_i^j}\}$: The voter i must prove that the each ciphertext c_{2i}^j is encrypted for p_i^j where $p_i^j \in [0, S]$. The process is as follows:

- For each ciphertext c_{2i}^j, voter i selects $\rho, h_k, e_k \in \mathbb{Z}_q$, where $k \in \{0, \ldots, p-1, p+1, \ldots, S\}$. $\{q_1, \ldots, q_{p-1}, q_{p+1}, \ldots, q_S\} = \{0, \ldots, p_i^j - 1, p_i^j + 1, \ldots, S\}$. Then, voter i calculates $f_k = \mathrm{pk}^{h_k} (\frac{c_{2i}^j}{g^{q_k}})^{e_k}$ and $f_p = \mathrm{pk}^\rho$. Calculate $\mathrm{ch} = H(f_1 \parallel \ldots \parallel f_S)$, $e_p = \mathrm{ch} - \sum_{k=[0,S], k \neq p} e_k$, and $h_p = \rho - r_i^j e_p$. The proof corresponding to c_{2i}^j is $\{\mathrm{ch}, \{h_\varsigma, e_\varsigma, f_\varsigma\}_{\varsigma \in [0,S]}\}$. Voter i needs to generate m proofs for all ciphertexts $\{c_{2i}^j\}_{j=1}^m$. The final result is $\pi_2 = \{\mathrm{ch}^j, \{h_\varsigma^j, e_\varsigma^j, f_\varsigma^j\}_{\varsigma \in [0,S]}\}_{j=1}^m$.

Voter i now needs to calculate $H(\pi_1 \parallel \pi_2)$ and submit it to the blockchain with the ballots. The two proofs $\{\pi_1, \pi_2\}$ are stored in the database. We summarize the proof generation phase as in Algorithm 2.

Algorithm 2 Proof generation

```
Require:
    Input: private keys $(x_i, y_i)$, all random numbers
    ($\{r_i^j\}_{j=1}^m, k_1, k_2, k_3, \rho^j, \{\{h_k^j, e_k^j\}_{k\in[0,S],k\neq p}\}_{j=1}^m$), the ciphertext $\{c_{2i}^j\}_{j=1}^m$, $\{q_1,\dots,q_{p-1},q_{p+1},\dots,q_S\} = \{0,\dots,p_i^j-1,p_i^j+1,\dots,S\}$ and ($g^{x_{i+1}}$, $g^{y_{i-1}}$);
    **Output**: $(\pi_1, \pi_2)$;
1: Compute $T_1 = g^{k_1}$, $T_2 = g^{k_2}$, $T_3 = (g^{x_{i+1}})^{k_1}\cdot(g^{y_{i-1}})^{k_2}\cdot g^{k_3}$;
2: Calculate challenge value $c = \mathrm{Hash}(T_1 \parallel T_2 \parallel T_3)$;
3: Compute $Z_1 = y_i c + k_1$, $Z_2 = -x_i c + k_2$;
4: **for** $j \leftarrow 1$ to $m$ **do**
5:     $Z_3^j = r_i^j c + k_3$;
6: **end for**
7: Let $\pi_1=\{T_1,T_2,T_3,Z_1,Z_2,\{Z_3^j\}_{j=1}^m\}$;
8: **for** $j \leftarrow 1$ to $m$ **do**
9:     **for** $(k \leftarrow 0$ to $S) \wedge (k \neq p)$ **do**
10:        $f_k^j = \mathrm{pk}^{h_k^j}(\frac{c_{2i}^j}{g^{q_k}})^{e_k^j}$;
11:    **end for**
12:    $f_p^j = \mathrm{pk}^{\rho^j}$;
13:    Calculate challenge value $\mathrm{ch}^j = H(f_1^j \parallel \dots \parallel f_S^j)$;
14:    Compute $e_p^j = \mathrm{ch}^j - \sum_{k=[0,S],k\neq p} e_k^j$ and $h_p^j = \rho^j - r_i^j e_p^j$;
15: **end for**
16: Let $\pi_2 = \{\mathrm{ch}^j, \{h_\varsigma^j, e_\varsigma^j, f_\varsigma^j\}_{\varsigma\in[0,S]}\}_{j=1}^m$;
17: **return** $(\pi_1, \pi_2)$.
```

4) Abortive case: If there are negative voters who do not vote or submit invalid votes, the smart contract will add these voters to the NVL and add the adjacent voters to the RVPVL. In this case, the abortive voters are removed from the list of voters. Then the administrator notifies their adjacent voters to help restore the voting process. Figure 3 presents the abortive model. The adjacent voters recalculate as follows:

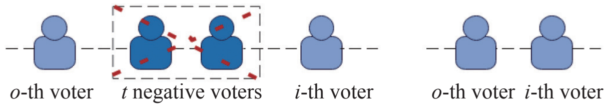


Figure 3 Abortive model.

- Suppose there are t negative voters after the o-th active voter and before the r-th active voter, voter o has voted $\{\{c_{1o}^j, c_{2o}^j\}_{j=1}^m, \pi_1, \pi_2\}$ in the voting phase. Without changing $\{c_{2o}^j\}_{j=1}^m$, voter o has to recalculate $\{c_{1o}^j\}_{j=1}^m$ using the public keys $(g^{y_{o-1}}, g^{x_r})$ and the same random numbers in $\{c_{2o}^j\}_{j=1}^m$, and we have

$$\{c_{1o}^j\}_{j=1}^m = g^{y_o x_r - x_o y_{o-1}}\cdot g^{r_o^j}$$

Then, the o-th voter must recalculate the proof π_1. The calculation process is the same as the voting phase and proof generation phase. The new ballots and the corresponding proof are $\{\{c_{1o}^{'j}, c_{2o}^j\}_{j=1}^m, \pi_1^{'}, \pi_2\}$. As there is no need to recalculate $\{c_{2o}^j\}_{j=1}^m$, the values of ballots have not been changed. The o-th voter recalculates the hash value of the two proofs, submits the ballots and the hash value on the blockchain, and stores the proofs in the database. The r-th voter also performs calculation similar to the o-th voter.

5) Vote verification: Before tallying ballots, all the ballots must be verified to ensure their correctness. One thing to verify is that all ballots are constructed using the correct parameters. The other thing to verify is that the value in every ballot is within a specific range. Let us take the example of verifying the ballots of v_i. It is worth noting that anyone can verify the ballots. In our scheme, the administrator is responsible for verifying all the ballots.

- Verify the construction: With the proof of $\pi_1 = \{T_1,T_2,T_3,Z_1,Z_2,\{Z_3^j\}_{j=1}^m\}$, anyone can verify if $c = \mathrm{Hash}(T_1 \parallel T_2 \parallel T_3)$, verify if $g^{Z_1} = T_1\cdot(g^{y_i})^c$, verify if $g^{Z_2} = T_2\cdot(g^{-x_i})^c$ and for $j\in[1,m]$, and verify if $(g^{x_{i+1}})^{Z_1}\cdot(g^{y_{i-1}})^{Z_2}\cdot g^{Z_3^j} = T_3\cdot(g^{y_i x_{i+1} - x_i y_{i-1}}\cdot g^{r_i^j})^c$.
- Verify value range: For $j\in[1,m]$, the verifier first checks $\mathrm{ch}^j \overset{?}{=} H(f_1^j \parallel f_2^j \parallel \dots \parallel f_S^j)$ and $\mathrm{ch}^j \overset{?}{=} \sum_{\varsigma=0}^S e_\varsigma^j$. Then, the verifier checks $f_\varsigma^j \overset{?}{=} \mathrm{pk}^{h_\varsigma^j}(\frac{c_{2i}^j}{g^{q_\varsigma}})^{e_\varsigma^j}$ where $q_\varsigma \in \{0,1,\dots,S\}$. Note that $\varsigma\in[0,S]$. The proof π_2 is accepted if these equations hold.

We summarize the vote verification phase as in Algorithm 3. Next we further analyze the correctness of the vote verification. If the voters honestly generate their votes, then the verification will definitely pass.

Algorithm 3 Vote verification

```
**Require:**
    **Input**: proofs $\{\pi_1, \pi_2\}$, the $i$-th voter's public keys $\{g^{y_i}, g^{x_i}\}$, the $i$-th voter's votes $\{c_{1i}^j, c_{2i}^j\}_{j=1}^m$, and the adjacent voters' public keys $\{g^{y_{i-1}}, g^{x_{i+1}}\}$;
    **Output**: Yes or No;
1: Verify if $c = \mathrm{Hash}(T_1 \parallel T_2 \parallel T_3)$;
2: Verify if $g^{Z_1} = T_1\cdot(g^{y_i})^c$;
3: Calculate the inverse element of $g^{x_i}$;
4: Verify if $g^{Z_2} = T_2\cdot(g^{-x_i})^c$;
5: **for** $j \leftarrow 1$ to $m$ **do**
6:     Verify if $(g^{x_{i+1}})^{Z_1}\cdot(g^{y_{i-1}})^{Z_2}\cdot g^{Z_3^j} = T_3\cdot(g^{y_i x_{i+1} - x_i y_{i-1}}\cdot g^{r_i^j})^c$;
7: **end for**
8: **for** $j \leftarrow 1$ to $m$ **do**
9:     Verify if $\mathrm{ch}^j \overset{?}{=} H(f_1^j \parallel \dots \parallel f_S^j)$;
10:    Verify if $\mathrm{ch}^j \overset{?}{=} \sum_{\varsigma=0}^S e_\varsigma^j$;
11:    **for** $\varsigma \leftarrow 0$ to $S$ **do**
12:        Verify if $f_\varsigma^j \overset{?}{=} \mathrm{pk}^{h_\varsigma^j}(\frac{c_{2i}^j}{g^{q_\varsigma}})^{e_\varsigma^j}$;
13:    **end for**
14: **end for**
15: If all the verification pass,
16: **return** Yes;
17: Otherwise
18: **return** No.
```

For the step 2 in Algorithm 3, we have

$$g^{Z_1} = g^{(y_i c + k_1)} = g^{y_i c} \cdot g^{k_1} = (g^{y_i})^c \cdot T_1$$

For the step 4 in Algorithm 3, we have

$$g^{Z_2} = g^{(-x_i c + k_2)} = g^{-x_i c} \cdot g^{k_2} = (g^{-x_i})^c \cdot T_2$$

For the step 6 in Algorithm 3, we have

$$\begin{aligned}
&(g^{x_{i+1}})^{Z_1} \cdot (g^{y_{i-1}})^{Z_2} \cdot g^{Z_3^j} \\
&= (g^{x_{i+1}})^{(y_i c + k_1)} \cdot (g^{y_{i-1}})^{(-x_i c + k_2)} \cdot g^{r_i^j c + k_3} \\
&= (g^{x_{i+1} y_i c}) \cdot g^{x_{i+1} k_1} \cdot (g^{-x_i c y_{i-1}}) \cdot (g^{y_{i-1} k_2}) \cdot g^{r_i^j c} \cdot g^{k_3} \\
&= (g^{x_{i+1} y_i} \cdot g^{-x_i y_{i-1}} \cdot g^{r_i^j})^c \cdot (g^{x_{i+1}})^{k_1} \cdot (g^{y_{i-1}})^{k_2} \cdot g^{k_3} \\
&= (g^{y_i x_{i+1} - x_i y_{i-1}} \cdot g^{r_i^j})^c \cdot T_3
\end{aligned}$$

For the step 12 in Algorithm 3, when $\varsigma \neq p$, the equation is obvious. When $\varsigma == p$, we have

$$\begin{aligned}
\mathrm{pk}^{h_p^j} \left(\frac{c_{2i}^j}{g^{q_p}} \right)^{e_p^j} &= \mathrm{pk}^{\rho^j - r_i^j e_p^j} \left(\frac{g^{p_i^j} \cdot \mathrm{pk}^{r_i^j}}{g^{q_p}} \right)^{e_p^j} \\
&= \mathrm{pk}^{\rho^j - r_i^j e_p^j} (\mathrm{pk}^{r_i^j})^{e_p^j} \\
&= \mathrm{pk}^{\rho^j} \\
&= f_p^j
\end{aligned}$$

6) Self-tallying phase: After all active voters submitted their ballots, the administrator publishes the private keys sk corresponding to the public key pk in the smart contract, and anyone can verify if $g^{\mathrm{sk}} = \mathrm{pk}$. We present the self-tallying phase here in two situations: One is the case without the abortive issue, and the other is the case with the abortive issue. Let us take the tally for the j-th candidate as an example.

- With all the ciphertexts $\{\{c_{1i}^j, c_{2i}^j\}_{i=1}^n\}_{j=1}^m$ and the secret key sk, as $\prod_{i=1}^n g^{y_i x_{i+1} - x_i y_{i-1}} = 1$, one can calculate $\prod_{i=1}^n c_{1i}^j = g^{\sum_{i=1}^n r_i^j}$, then calculate $(g^{\sum_{i=1}^n r_i^j})^{sk} = \mathrm{pk}^{\sum_{i=1}^n r_i^j}$, and calculate $\prod_{i=1}^n c_{2i}^j = g^{\sum_{i=1}^n p_i^j} \cdot \mathrm{pk}^{\sum_{i=1}^n r_i^j}$. The final poll received by the j-th candidate is $\frac{g^{\sum_{i=1}^n p_i^j} \cdot \mathrm{pk}^{\sum_{i=1}^n r_i^j}}{\mathrm{pk}^{\sum_{i=1}^n r_i^j}} = g^{\sum_{i=1}^n p_i^j} = g^{p^j}$. After calculating the ballots for each candidate, we have $\{g^{p^j}\}_{j=1}^m$. The $\{p^j\}_{j=1}^m$ can be derived by brute-force search from a relatively small value set.
- We use avn to denote the number of the rest active voters. When there are negative voters in the voting process, we have ciphertexts as $\{\{c_{1k}^j, c_{2k}^j\}_{k=1}^{\mathrm{avn}}\}_{j=1}^m$. For the new active voters list, we have $\prod_{k=1}^{\mathrm{avn}} g^{y_k x_{k+1} - x_k y_{k-1}} = 1$. One can calculate $\prod_{k=1}^{\mathrm{avn}} c_{1k}^j = g^{\sum_{k=1}^{\mathrm{avn}} r_k^j}$, then calculate $(g^{\sum_{k=1}^{\mathrm{avn}} r_k^j})^{\mathrm{sk}} = \mathrm{pk}^{\sum_{k=1}^{\mathrm{avn}} r_k^j}$, and calculate $\prod_{k=1}^{\mathrm{avn}} c_{2k}^j = g^{\sum_{k=1}^{\mathrm{avn}} p_k^j} \cdot \mathrm{pk}^{\sum_{k=1}^{\mathrm{avn}} r_k^j}$. The final poll received by the j-th candidate is $\frac{g^{\sum_{k=1}^{\mathrm{avn}} p_k^j} \cdot \mathrm{pk}^{\sum_{k=1}^{\mathrm{avn}} r_k^j}}{\mathrm{pk}^{\sum_{k=1}^{\mathrm{avn}} r_k^j}} = g^{\sum_{k=1}^{\mathrm{avn}} p_k^j} = g^{p^j}$.

After calculating the ballots for each candidate, we have $\{g^{p^j}\}_{j=1}^m$. The $\{p^j\}_{j=1}^m$ can be derived by brute-force search from a relatively small value set. We summarize the self-tallying phase as in Algorithm 4 and Algorithm 5.

Algorithm 4 Self-tallying (without abortive issue)

```
Require:
    Input: the secret keys sk of the candidates, all the ciphertexts $\{\{c_{1i}^j, c_{2i}^j\}_{i=1}^n\}_{j=1}^m$;
    Output: $\{g^{p^j}\}_{j=1}^m$;
1: for $j \leftarrow 1$ to $m$ do
2:     Calculate $\prod_{i=1}^n c_{1i}^j = g^{\sum_{i=1}^n r_i^j}$;
3:     Calculate $(g^{\sum_{i=1}^n r_i^j})^{sk} = \mathrm{pk}^{\sum_{i=1}^n r_i^j}$;
4:     Calculate $\prod_{i=1}^n c_{2i}^j = g^{\sum_{i=1}^n p_i^j} \cdot \mathrm{pk}^{\sum_{i=1}^n r_i^j}$;
5:     Calculate $\frac{g^{\sum_{i=1}^n p_i^j} \cdot \mathrm{pk}^{\sum_{i=1}^n r_i^j}}{\mathrm{pk}^{\sum_{i=1}^n r_i^j}} = g^{\sum_{i=1}^n p_i^j} = g^{p^j}$;
6: end for
7: return $\{g^{p^j}\}_{j=1}^m$.
```

Algorithm 5 Self-tallying (with abortive issue)

```
Require:
    Input: the secret keys sk of the candidates, all the ciphertexts $\{\{c_{1k}^j, c_{2k}^j\}_{k=1}^{\mathrm{avn}}\}_{j=1}^m$;
    Output: $\{g^{p^j}\}_{j=1}^m$;
1: for $j \leftarrow 1$ to $m$ do
2:     Calculate $\prod_{k=1}^{\mathrm{avn}} c_{1k}^j = g^{\sum_{k=1}^{\mathrm{avn}} r_k^j}$;
3:     Calculate $(g^{\sum_{k=1}^{\mathrm{avn}} r_k^j})^{sk} = \mathrm{pk}^{\sum_{k=1}^{\mathrm{avn}} r_k^j}$;
4:     Calculate $\prod_{k=1}^{\mathrm{avn}} c_{2k}^j = g^{\sum_{k=1}^{\mathrm{avn}} p_k^j} \cdot \mathrm{pk}^{\sum_{k=1}^{\mathrm{avn}} r_k^j}$;
5:     Calculate $\frac{g^{\sum_{k=1}^{\mathrm{avn}} p_k^j} \cdot pk^{\sum_{k=1}^{\mathrm{avn}} r_k^j}}{pk^{\sum_{k=1}^{\mathrm{avn}} r_k^j}} = g^{\sum_{k=1}^{\mathrm{avn}} p_k^j} = g^{p^j}$;
6: end for
7: return $\{g^{p^j}\}_{j=1}^m$.
```

VI. Security Analysis

This section is devoted to a theoretical security analysis of our protocol. The following assumptions are used in our security analysis.

- All voters will protect their privacy and not disclose their confidential information.
- The Exponential ElGamal Cryptosystem and the Group Encryption Scheme are secure.
- All content submitted to the blockchain is trusted and secure.
- The administrator cannot simultaneously collude with two voters separated by one voter in the sorted voter's list.
- The administrator cannot collude with the last voter in the voters' list sorted by the smart contract.

Theorem 1 If the decisional Diffie-Hellman (DDH) assumption is hard, then our scheme can protect the voters' secrecy.

Proof Suppose there exists a PPT adversary $\mathscr{A}$ who can break the security of our scheme, then we can build a PPT algorithm $\mathscr{C}$ to break the DDH assumption. We define three games $\mathrm{Game}_0, \mathrm{Game}_1$, and Game_2 in the proof. The advantage for the adversary $\mathscr{A}$ to win in Game_i is denoted as $\mathrm{Adv}_{\mathscr{A},\mathrm{Game}_i}$.

Game_0: $\mathscr{A}$ sends two different evaluation points (p_0, p_1) to $\mathscr{C}$ as challenge. Then $\mathscr{C}$ randomly selects $b \in \{0,1\}$ and generates one ballot as a challenge, which is denoted as (c_1, c_2). $\mathscr{A}$ outputs a guess b'. We have $\mathrm{Adv}_{\mathscr{A},\mathrm{Game}_0} = \Pr[b = b'] - \frac{1}{2}$.

Game_1: Let the DDH instance be (g^a, g^b, Γ), where $\Gamma \in \{g^{ab}, g^{\mu}\}$ and $\mu \in_R \mathbb{Z}_p^*$. Set the public key of the administrator as g^a. Let $c_1^* = g^b$ and $c_2^* = g^{p_b} g^{\Gamma}$. With $(c_1, c_2) = (g^{y_i x_{i+1} - x_i y_{i-1}} \cdot g^r, g^{p_b} \cdot \mathrm{pk}^r)$, (c_1, c_2) and (c_1^*, c_2^*) are indistinguishable from $\mathscr{A}$'s view. This is because r and b are uniformly and randomly selected from $\mathbb{Z}_p^*$, (c_1, c_2) and (c_1^*, c_2^*) have exactly the same distribution. We have $|\mathrm{Adv}_{\mathscr{A},\mathrm{Game}_1} - \mathrm{Adv}_{\mathscr{A},\mathrm{Game}_0}| = 0$.

Game_2: Based on Game_1, we have $c_1^* = g^b$ and $c_2^* = g^{p_b} g^{\Gamma}$. The $\mathscr{A}$ outputs a guess b''. If $b = b''$, we can make use of $\mathscr{A}$ to solve the DDH assumption. We have $|\mathrm{Adv}_{\mathscr{A},\mathrm{Game}_2} - \mathrm{Adv}_{\mathscr{A},\mathrm{Game}_1}| = \backepsilon_{\mathrm{DDH}}$. Therefore, $|\mathrm{Adv}_{\mathscr{A},\mathrm{Game}_2} - \mathrm{Adv}_{\mathscr{A},\mathrm{Game}_0}| = \backepsilon_{\mathrm{DDH}}$. Thus the probability for $\mathscr{A}$ to win in the game is $\frac{1}{2} + \backepsilon_{\mathrm{DDH}}$.

Theorem 2 The final voting result can be calculated by any entity in the end of election using the administrator's private key.

Proof Take the candidate j as an example. After voting, the administrator must store his/her private key sk in the smart contract. As the public key pk is already published before voting, anyone can use it to check the correctness of sk. In our protocol, as $\prod_{i=1}^{n} g^{y_n x_1 - x_n y_{n-1}} = 1$, by multiplying all the votes corresponding to candidate j, one can obtain encryption result of the final score g^{p^j} of candidate j, namely

$$\left(\prod_{i=1}^{n} c_{1i}^{j} = g^{\sum_{i=1}^{n} r_i^j}, \prod_{i=1}^{n} c_{2i}^{j} = g^{\sum_{i=1}^{n} p_i^j} \cdot \mathrm{pk}^{\sum_{i=1}^{n} r_i^j}\right)$$

Using the private key sk, one can decrypt the above ciphertext to obtain g^{p^j}. Finally, p^j can be easily calculated using brute-force computation. All the content required for the process is public, so anyone can calculate the final voting result.

Theorem 3 The election cannot be aborted by any voter refusing to submit their votes or submit invalid votes.

Proof All the voters who do not vote or submit invalid ballots will be added in the NVL and they cannot vote anymore. For resuming the election, the administrator adds adjacent voters to the RVPVL and notifies them to follow Algorithm 3. The final voting result can be calculated as presented in Algorithm 4 or Algorithm 5.

Theorem 4 The final voting result cannot be calculated before the end of the voting phase.

Proof During the voting phase, the last voter who did not vote can only get the following results before voting:

$$\left(\prod_{i=1}^{n} c_{1i}^{j} = g^{\sum_{i=1}^{n} r_i^j}, \prod_{i=1}^{n} c_{2i}^{j} = g^{\sum_{i=1}^{n} p_i^j} \cdot \mathrm{pk}^{\sum_{i=1}^{n} r_i^j}\right)_{j=1}^{m}$$

Each result is encrypted by the ElGamal encryption. As long as the ElGamal cryptosystem is secure, it is impossible for voters to obtain intermediate results without the administrator's private key sk.

Theorem 5 Only eligible voters can vote.

Proof Before voting, each voter must prove their identity to the administrator and the identity information of each voter is published on the blockchain. All the voters' addresses are submitted to the smart contract by the administrator. The voters' addresses will be added to the AVL and the smart contract can control that only voters in the AVL can vote.

Theorem 6 Once content is submitted and stored on the public bulletin board, it cannot be modified or deleted.

Proof Due to the immutable and transparent nature of Ethereum itself, all content submitted on the blockchain cannot be modified and can be accessed by anyone.

Theorem 7 Each voter can only vote once, and excess ballots will be rejected.

Proof After a voter votes and the ballots are verified to be valid, the corresponding voter's address will be added to the HVAVL and deleted from the NVAVL. Only voters whose addresses are in the NVAVL are allowed to vote, which is automatically controlled by the smart contract.

Theorem 8 Invalid ballots can be detected by any entity.

Proof Besides the ballots, each voter has to provide the zero-knowledge proof in proof generation phase. Only the well-formed ballots can pass the verification, which guarantees that the voter has to calculate honestly.

Theorem 9 Each voter is able to verify whether the ballots are published and tallied correctly and any entity can verify the final voting result.

Proof As all the ballots and corresponding proof are stored in the blockchain or the database, each voter can check if the ballots are published correctly. Since our protocol satisfies self-tallying property, anyone can verify the voting result by recalculating it using Algorithm 4 or Algorithm 5. Active voters can check whether their ballots are used to calculate the final result.

VII. Performance Analysis

In this section, we have conducted theoretical and experimental analysis of the efficiency of our protocol from six aspects: voting calculation time, vote size, proof generation time, proof size, and voting verification time, voting result calculation time. In addition, we compared our protocol with the works [10], [12] closed to ours in the six aspects described above.

When calculating the costs of voters in PriScore [12], we add together the costs of voters in the commitment stage and the voting stage, as the commitment stage is required to address the abortive issue. In the protocol [10] proposed by Yang *et al.*, most of the calculation is done by a trusted SGX. With SGX, there is no proof of ballots and the verification is not required either. We regard the part that SGX helps voters calculate as costs for voters. All the experiments were completed on a desktop computer with the following specifications: Intel(R) Core(TM)i5-10400F CPU @ 2.90 GHz with 12 MB shared L3 cache, 16 GB of 2400 MHz memory.

We use t_e to denote the computation time of one exponentiation, where $t_e \approx 2.325$ ms, $t_\times$ to denote the computation time of multiplication operation, where $t_\times \approx 0.012$ ms, s_e to denote the size of a group element, where $s_e = 8$ bytes, and s_r to denote the size of a random number in $\mathbb{Z}_p$, where $s_r = 8$ bytes.

1. Voting calculation time

According to the Algorithm 1, each score p_i^j is encrypted as

$$E(p_i^j, r_i^j, \text{pk}, x_i, y_i, g^{x_{i+1}}, g^{y_{i-1}}) = (g^{y_i x_{i+1} - x_i y_{i-1}} \cdot g^{r_i^j}, g^{p_i^j} \cdot \text{pk}^{r_i^j})$$

We use T_{votes} to denote the total time each voter spends calculating their votes and m to denote the number of candidates here. For our work, we have

$$T_{\text{votes}} = [(3t_e + 2t_\times) + (2t_e + t_\times)]m = 5t_e m + 3t_\times m$$

With $m = 1, 5, 10, 15, 20$, the result of testing T_{votes} is shown in Figure 4(a). Compared to the protocols in [10] and [12], the vote generation time of each voter in our system is much shorter.

2. Vote size

We use S_{votes} to denote the total size of each voter's ballots and m to denote the number of candidates here. In our work, we have

$$S_{\text{votes}} = 2s_e m$$

The test of S_{votes} is shown in Figure 4(b). The vote size of each voter in our protocol is nearly equal to the vote size in [12] and almost twice the vote size in [10].

3. Proof generation time

We use T_{proof} to denote the total time spent by one voter on proof generation. According to the Algorithm 2, we have

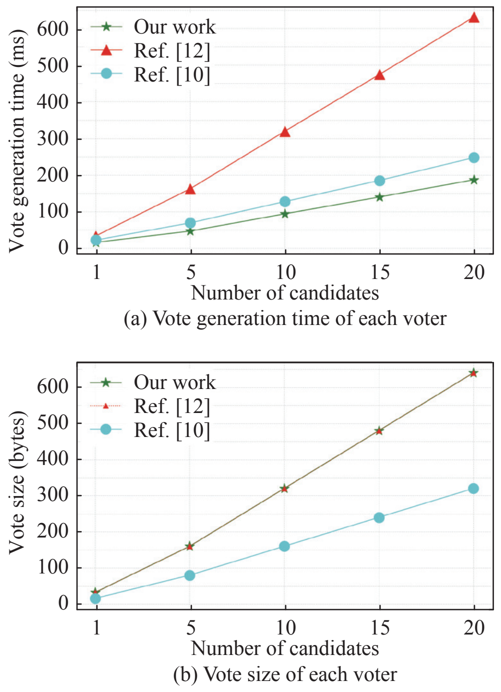


Figure 4 Voting test.

$$T_{\text{proof}} = (5 + 2mS + m)t_e + 2Smt_\times$$

The proof generation time T_{proof} in our system is related to two variables, namely the maximum score S and the number of candidates m. With $m = 1, 5, 10, 15, 20$, we test T_{proof} and the result is shown in Figure 5(a). With $S = 5, 10, 15, 20, 25$, we test T_{proof} and the result is shown in Figure 5(b). What's exciting is that the time of generating proof in our protocol is much shorter than the corresponding time spent in PriScore.

4. Proof size

We use S_{proof} to denote the size of the proof generated by each voter. According to Algorithm 2, we have

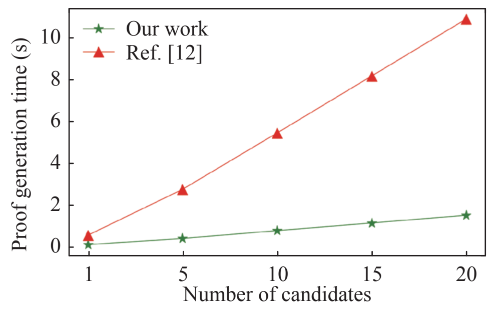
$$S_{\text{proof}} = (S + 2)s_e m + 2s_r m(S + 1)$$

The proof size S_{proof} in our protocol is related to two variables, namely the maximum score S and the number of candidates m. With $m = 1, 5, 10, 15, 20$, we test S_{proof} and the result is shown in Figure 6(a). With $S = 1, 5, 10, 15, 20$, we test T_{proof} and the result is shown in Figure 6(b). The size of proof in our scheme is very small compared to the proof size in PriScore. However, compared to the size of the ballots, it is still larger, so we store the proof in the database to reduce the cost of storing data in the blockchain.

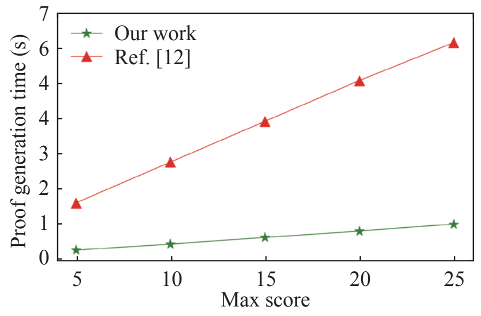
5. Voting verification time

We use T_{verify} to denote the time required to verify each voter's ballots. According to Algorithm 3 in Section V, we have

$$T_{\text{verify}} = (2mS + 4m + 6)t_e + (2mS + 5m + 2)t_\times$$

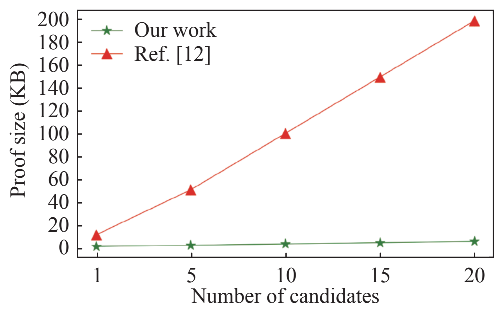


(a) Comparison of proof generation time based on the number of candidates

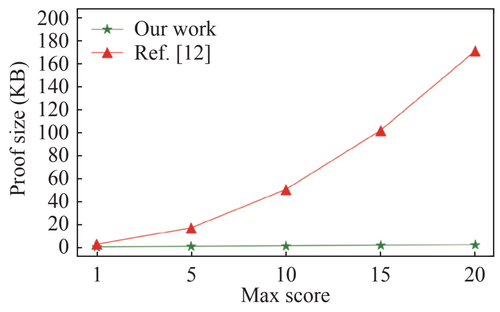


(b) Comparison of proof generation time basd on the max score

Figure 5 Proof generation time test.



(a) Comparison of proof size based on the number of candidates



(b) Comparison of proof size based on the max score

Figure 6 Proof size test.

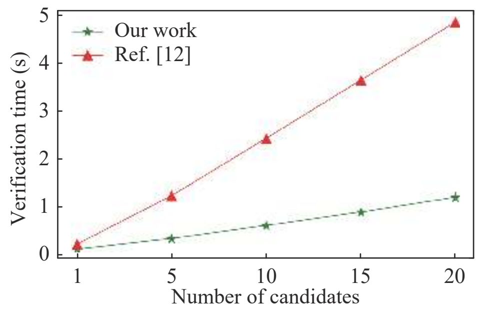
The proof size T_{verify} in our scheme is related to two variables, namely the maximum score S and the number of candidates m. We first test T_{verify} with $m = 1, 5, 10, 15, 20$ and the result is shown in Figure 7(a). Then, with $S = 5, 10, 15, 20, 25$, we test T_{verify} and the result is shown in Figure 7(b). The time of verification in our protocol is nearly one quarter of that in PriScore.

6. Time of calculating voting result

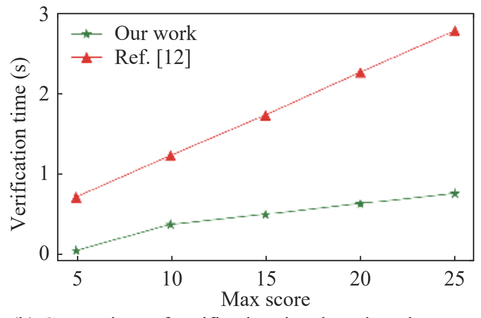
We use T_{cr} to denote the time one takes to calculate the voting result and n to denote the number of voters. According to Algorithm 4, we have

$$T_{cr} = 2mt_e + 2mnt_\times$$

With $n = 50, 500, 1000, 10000, 100000$, we test T_{cr} and the result is shown in Figure 8. When the number of voters reaches 100000, the calculation time is 10.815 s, compared to 10.476 s in [12], and 5.021 s in [10].



(a) Comparison of verification time based on the number of candidates



(b) Comparison of verification time based on the max score

Figure 7 Verification test.

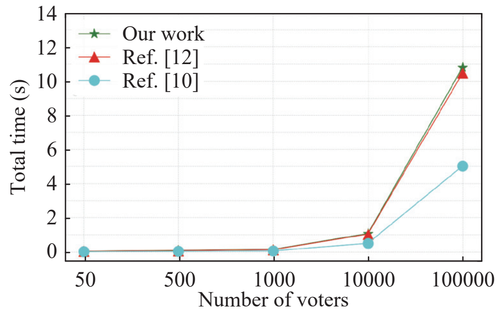


Figure 8 Time of calculating voting result test.

VIII. Conclusion and Future Work

This paper presents a secure self-tallying e-voting protocol where every step of voting and the phase of tallying ballots can be verified by any party. Each voter can score candidates within a certain range. Each ballot can be verified using the proof generated by the corresponding voter. By combining Exponential ElGamal Cryptosystem and distributed encryption, the adaptive issue and the abortive issue are efficiently solved. Performance analysis indicates that our protocol is suitable for large-scale election as the low computational overhead.

But there are some limitations and weaknesses in our scheme. When the administrator colludes with adjacent voters of the i-th voter, the voting content of the i-th voter can be restored, and when the administrator colludes with the last voter, they can obtain the voting results in advance. In addition, we do not consider the situation where some voters are forced to disclose their secret information to the adversary, in which the adversary can obtain all the voting contents of the coerced voters. The coercion issue has been studied by some excellent works [33]–[35]. Achenbac *et al.* [35] adopted deniable revoting to realize coercion-resistant and Lueks *et al.* [33] and Locher *et al.* [34] utilized the technology of cryptographic shuffle to solve the coercion issue. None of them can simultaneously achieve self-tallying property and coercion-resistant. We will explore the possibility of achieving it in the future.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant No. U20012052), the Foundation of Science and Technology on Communication Security Laboratory of China (Grant No. 61421030108022110), the Open Fund of the State Key Laboratory of Integrated Services Networks, Xidian University (Grant No. ISN23-23), and the Key Research and Development Program of Shaanxi (Grant No. 2023-YB-GY-214).

References

[1] Z. Q. Chen, L. X. Jiang, and C. Q. Li, "Label augmented and weighted majority voting for crowdsourcing," *Information Sciences*, vol. 606, pp. 397–409, 2022.

[2] N. Jiang, D. Xu, J. Zhou, *et al.*, "Toward optimal participant decisions with voting-based incentive model for crowd sensing," *Information Sciences*, vol. 512, pp. 1–17, 2020.

[3] P. Drotár, M. Gazda, and L. Vokorokos, "Ensemble feature selection using election methods and ranker clustering," *Information Sciences*, vol. 480, pp. 365–380, 2019.

[4] L. Chamakura and G. Saha, "An instance voting approach to feature selection," *Information Sciences*, vol. 504, pp. 449–469, 2019.

[5] J. H. Zhai, X. Z. Wang, and X. H. Pang, "Voting-based instance selection from large data sets with MapReduce and random weight networks," *Information Sciences*, vol. 367-368, pp. 1066–1077, 2016.

[6] J. W. Cao, Z. P. Lin, G. B. Huang, *et al.*, "Voting based extreme learning machine," *Information Sciences*, vol. 185, no. 1, pp. 66–77, 2012.

[7] X. S. Wang, Y. Y. Gu, Y. H. Cheng, *et al.*, "An ensemble classifier based on selective independent component analysis of DNA microarray data," *Chinese Journal of Electronics*, vol. 18, no. 4, pp. 643–649, 2009.

[8] T. C. Song, J. Feng, S. Li, *et al.*, "Color context binary pattern using progressive bit correction for image classification," *Chinese Journal of Electronics*, vol. 30, no. 3, pp. 471–481, 2021.

[9] A. Kiayias and M. Yung, "Self-tallying elections and perfect ballot secrecy," in *Public Key Cryptography*, Springer, Berlin, pp. 141–158, 2002.

[10] X. C. Yang, X. Yi, A. Kelarev, *et al.*, "A distributed networked system for secure publicly verifiable self-tallying online voting," *Information Sciences*, vol. 543, pp. 125–142, 2021.

[11] X. C. Yang, X. Yi, S. Nepal, *et al.*, "Decentralized voting: A self-tallying voting system using a smart contract on the ethereum blockchain," in *Web Information Systems Engineering–WISE 2018*, Springer, Cham, pp. 18–35, 2018.

[12] Y. Yang, Z. S. Guan, Z. G. Wan, *et al.*, "Priscore: Blockchain-based self-tallying election system supporting score voting," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4705–4720, 2021.

[13] Y. K. Lin and P. Zhang, "Blockchain-based complete self-tallying e-voting protocol," in *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, Lanzhou, China, pp. 47–52, 2019.

[14] B. Yu, J. K. Liu, A. Sakzad, *et al.*, "Platform-independent secure blockchain-based voting system," in *Information Security*, Springer, Cham, pp. 369–386, 2018.

[15] M. Chaieb, S. Yousfi, P. Lafourcade, *et al.*, "Verify-your-vote: A verifiable blockchain-based online voting protocol," in *Information Systems*, Springer, Cham, pp. 16–30, 2019.

[16] J. Huang, D. B. He, Y. T. Chen, *et al.*, "A blockchain based self-tallying voting protocol with maximum voter privacy," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 5, pp. 3808–3820, 2022.

[17] P. Tarasov and H. Tewari, "Internet voting using Zcash," *IACR Cryptol. ePrint Arch.*, vol. 2017, article no. 585, 2017.

[18] G. X. Zeng, M. Q. He, and S. M. Yiu, "A secure and self-tallying E-voting system based on blockchain," in *Information Security Applications*, You, I., Ed. Springer, Cham, pp. 67–76, 2020.

[19] Y. N. Li, W. Susilo, G. M. Yang, *et al.*, "A blockchain-based self-tallying voting protocol in decentralized IoT," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 119–130, 2022.

[20] G. X. Zeng, M. Q. He, S. M. Yiu, *et al.*, "Corrigendum to: A self-tallying electronic voting based on blockchain," *The Computer Journal*, vol. 66, no. 2, pp. 523–523, 2023.

[21] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in *Financial Cryptography and Data Security*, Springer, Cham, pp. 357–375, 2017.

[22] I. Stančíková and I. Homoliak, "SBvote: Scalable self-tallying blockchain-based voting," in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, Tallinn, Estonia, pp. 203–211, 2023.

[23] J. Dossogne and F. Lafitte, "Blinded additively homomorphic encryption schemes for self-tallying voting," in *Proceedings of the 6th International Conference on Security of Information and Networks*, Aksaray, Turkey, pp. 173–180, 2013.

[24] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[25] D. L. Xu, W. Shi, W. S. Zhai, *et al.*, "Multi-candidate voting model based on blockchain," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 12, pp. 1891–1900, 2021.

[26] F. Javani and A. T. Sherman, "BVOT: Self-tallying boardroom voting with oblivious transfer," *Information Security Journal: A Global Perspective*, vol. 33, no. 1, pp. 42–53, 2024.

[27] W. Y. Xue, Y. Yang, Y. L. Li, *et al.*, "ACB-vote: Efficient, flexible, and privacy-preserving blockchain-based score voting with anonymously convertible ballots," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. , pp. 3720–3734, 2023.

[28] F. Z. Chentouf and S. Bouchkaren, "Security and privacy in smart city: A secure e-voting system based on blockchain," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 13, no. 2, pp. 1848–1857, 2023.

[29] A. Singh, A. Ganesh, R. R. Patil, *et al.*, "Secure voting web site using ethereum and smart contracts," *Applied System In-*

novation, vol. 6, no. 4, article no. 70, 2023.

[30] N. Jaiswar, S. Deodhar, H. Gupta, *et al.*, “E -voting system using blockchain,” *International Journal for Research in Applied Science & Engineering Technology*, vol. 11, no. 4, pp. 2090–2095, 2023.

[31] A. C. Naik, A. M. Prajapati, S. N. Pandey, *et al.*, “Blockchain based E-voting system,” in *2023 7th International Conference on Trends in Electronics and Informatics (ICOEI)*, Tirunelveli, Inida, pp. 316–320, 2023.

[32] C. P. Schnorr, “Efficient signature generation by smart cards,” *Journal of Cryptology*, vol. 4, pp. 161–174, 1991.

[33] W. Lueks, I. Querejeta-Azurmendi, and C. Troncoso, “VoteAgain: A scalable coercion-resistant voting system,” in *Proceedings of the 29th USENIX Security Symposium*, Boston, USA, pp. 1553-1570, 2020.

[34] P. Locher, R. Haenni, and R. E. Koenig, “Coercion-resistant internet voting with everlasting privacy,” in *Financial Cryptography and Data Security*, Springer, Berlin, pp. 161–175, 2016.

[35] D. Achenbach, C. Kempka, C. Kempka, *et al.*, “Improved Coercion-Resistant electronic elections through deniable Re-Voting,” *The USENIX Journal of Election Technology and Systems*, vol. 3, no. 2, pp. 26–45, 2015.

Jingyu YAO is currently an M.S. candiate in software engineering with the School of Computer Science, Shaanxi Normal University, Xi’an, China. His research interests include cryptography and information security.
(Email: jingyuyao@snnu.edu.cn)

Bo YANG received the Ph.D. degree in cryptography from Xidian University, Xi’an, China, in 1999. He is currently a Professor with the School of Computer Science, Shaanxi Normal University, Xi’an, China. His research interests include information security and cryptography.
(Email: byang@snnu.edu.cn)

Tao WANG received the Ph.D. degree in computer science and technology from Northwestern Polytechnical University, Xi’an, China, in 2012. He is currently an Associate Professor with the School of Computer Science, Shaanxi Normal University, Xi’an, China. He is also a Research Fellow with the State Key Laboratory of Integrated Services Networks, Xidian University, Xi’an, China, and the Science and Technology on Communication Security Laboratory, Chengdu, China. His research interests include cryptography, blockchain, and information security.
(Email: water@snnu.edu.cn)

Wenzheng ZHANG is currently a Research Fellow with the Science and Technology on Communication Security Laboratory, Chengdu, China. His research interests include information security and cryptography.
(Email: zwz85169038@sina.com)