Decentralized Voting: A Self-tallying Voting System Using a Smart Contract on the Ethereum Blockchain

Xuechao Yang[1(✉)], Xun Yi[1], Surya Nepal[2], and Fengling Han[1]

[1] School of Science, RMIT University, Melbourne, VIC 3000, Australia
xuechao.yang@rmit.edu.au
[2] CSIRO Data61, Sydney, NSW 2122, Australia

Abstract. Electronic online voting has been piloted in various countries in the recent past. These experiments show that further research is required, to improve the security guarantees of such systems, in terms of vote confidentiality and integrity and validity verification. In this paper we argue that blockchain technology, combined with modern cryptography can provide the transparency, integrity and confidentiality required from reliable online voting. Furthermore, we present a decentralized online voting system implemented as a smart contract on the Ethereum blockchain. The system has no hardwired restrictions on possible vote assignments to candidates, protects voter confidentiality by using a homomorphic encryption system and stores proofs for each element of a vote. To the best of our knowledge, our proposed system is the first decentralized ranked choice online voting system in existence. The underlying Ethereum platform enforces the correct execution of the voting protocol. We also present a security and performance analysis, showing the feasibility of our proposed protocol for real-world voting applications at large scale.

Keywords: Decentralized voting · Ethereum blockchain
Smart contract · Self-tallying

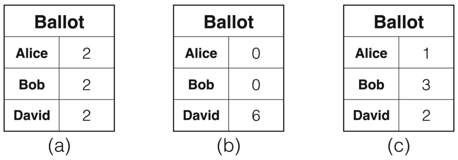
1 Introduction

A blockchain is a public, append-only, immutable ledger maintained by a decentralised peer-to-peer network. Whilst first designed for digital currencies without trusted third parties, blockchain technology has now moved into many fields beyond finance.

In this paper, we focus on blockchain-based online voting. There are a number of existing proposals for such a system, using the blockchain as a public bulletin board to store the voting data, such as FollowMyVote [7] and TIVI [21]. These proposals achieve voter privacy by involving trusted authorities that obfuscate the relation between real-world identities and keys [7], or by shuffling encrypted votes before decrypting [21].

We propose a self-tallying online voting system using a smart contract deployed on Ethereum. The system reduces the responsibilities of election authorities to a minimum and allows candidate ranking, instead of just voting for one candidate [14]. The system's voting mechanism is inspired by score voting [23], which enables voters to assign points to different candidates directly without any restrictions apart from the total number of available points specified (Fig. 1).

In most online voting systems, a tallying authority tallies the votes and decrypts the result [1,22]. Self-tallying was introduced by Kiayisa and Yung [10] and developed by Groth [9], which converts tallying into an open procedure, that allows any voter or a third-party observer to perform the tally computation once all votes are cast. Unfortunately, self-tallying protocols have a fairness drawback, as the last voter can compute the tallying before anyone else. McCorry et al. [14] proposed a self-tallying protocol that avoids this adaptive issue. However, the system requires voters to vote in two rounds. Our system does not have any adaptive issue with only one round of voting.



Ballot	
Alice	2
Bob	2
David	2

(a)

Ballot	
Alice	0
Bob	0
David	6

(b)

Ballot	
Alice	1
Bob	3
David	2

(c)

Fig. 1. In this example six points can be distributed amongst candidates.

Online privacy preservation is one of the most pressing concerns in Internet technology [11–13,16–20]. Therefore, maintaining the privacy and security of voters is the priority for our online voting system. Our proposed decentralized voting system uses the exponential ElGamal encryption system [3] and an open vote network protocol [14]. The additive homomorphism property of the cryptographic system makes it possible to tally encrypted votes directly without decrypting them. Our proposed system also incorporates cryptographic proofs to ensure the integrity of the voting process and to verify the validity of each vote before it is saved to the blockchain. To the best of our knowledge our voting system is the first decentralized ranked choice online voting system in existence, which meets the following security requirements [22,23]:

Eligibility of Voters: Only authorized voters can submit their cast votes.

Multiple-Voting Detection: Multiple voting by any one voter is detected and identified.

Privacy of Voters: All submitted votes must be stored securely and secretly and should not reveal voting preferences of the voters.

Integrity of Ballot: No one can modify or duplicate any submitted votes.

Correctness of Tallied Result: Only verified votes are counted to calculate the result.

End-to-End Voter Verifiable: Every voter is able to verify whether his/her vote is posted and counted correctly, and also able to verify the eligibility of other submissions.

The rest of this paper is organized as follow: Sect. 2 demonstrates a simple voting contract that is deployed on Ethereum. Section 3 describes the cryptographic models used in our online voting system. Section 4 presents our proposed online voting system. A security and performance analysis can be found in Sects. 5 and 6, respectively. Finally, Sect. 7 concludes the paper.

2 Preliminaries on Smart Contract Voting

Blockchain-Based Smart Contracts: Blockchain technology was first introduced by the Bitcoin digital currency in 2008 [2]. Bitcoin proposed a solution to securely maintain a decentralized ledger in the presence of a Byzantine failure model [4], in which nodes may act maliciously.

Blockchain ledgers were originally designed to record monetary transactions, but the concept has been widened to provide support for general purpose computing. Ethereum [5] provides a Turing complete platform for decentralized smart contracts. Smart Contracts were first described in 1996 by Nick Szabo [6] and are autonomously executing contracts written in computer code.

The Blockchain provides the following properties which make smart contracts possible:

Transparency. Blockchain transactions are public and can be verified by anyone.

Immutability. The transaction history of a blockchain cannot be altered. As such, the Blockchain can be seen as an append-only database.

Trustless. Participants in blockchain transactions do not have to rely on a trusted third party for their interactions. Trust is provided by the underlying consensus protocol.

Decentralized Voting: Ethereum provides a natural platform for our distributed voting system, in that it provides a decentralized "public bulletin board" to support coordination amongst voters. The execution of the election procedure is enforced by the same consensus mechanisms that secures the Blockchain. The smart contract code is stored on the Blockchain and executed by all peers to reach consensus on its output.

We present a simple voting contract written in Ethereum's Solidity language. The implementation was deployed on Ethereum's Kovan test network and the contract's interface is as follows:

VotingContract(candidateList, voterList, definedPoint): this is the constructor function of the contract. In order for the election administrator to deploy a new contract, there are three parameters that have to be provided: (1) a list of candidates; (2) a list of eligible voters; and (3) total available points. Once the contract is deployed, it is immutable.

submitVote(vote, voterSign): an eligible voter is able to cast and submit a vote via this function. This function calls a contract internal **verifyPendingVote**(vote) function, which verifies the eligibility of the vote. The function returns true (success) or false.
verifyAddedVote(voterID) constant returns (bool): Each voter is able to verify the eligibility of any other voter's submission before self-tallying.
tallyVotes(candidateName) constant returns (uint8): voters can tally any candidate's final received points independently by using this self-tallying function.

The above voting contract submits and stores data in plaintext format. In order to protect the privacy of voters, an encryption system has to be used.

3 Preliminaries on Cryptography

In this section, we introduce the underlying cryptographic building blocks for our proposed online voting system. We combine two cryptographic systems to ensure both voter privacy and verifiability of the result. The two systems involved are the ElGamal Cryptosystem [3] and the distributed encryption protocol described in [8].

ElGamal Cryptosystem: We assume that the cyclic group (G, q, g) is defined. A user has a public key y and private key x. The ElGamal cryptosystem consists of the following algorithms:

Encryption. To encrypt a plaintext message $m \in G$: Randomly choose an integer r from $\mathbb{Z}_q^*$; Computes $c_1 = g^r$; Computes $c_2 = g^m \cdot y^r$. And the encrypted message can be presented as $E(m) = (c_1, c_2)$.

Decryption. A user computes and broadcasts a partially decrypted value, and the final plaintext is revealed. For the ciphertext (c_1, c_2), decryption proceeds as follows: The user with secret key x computes ${c_1}^x$ and broadcasts to others; Everyone is able to compute $\frac{c_2}{{c_1}^x} = g^m$. Finally, m can be revealed by computing a discrete logarithm.

Homomorphism. ElGamal encryption has an inherited homomorphic property [23], which allows multiplication and exponentiation to be performed on a set of ciphertexts without decrypting them, such as $E(m_1) \times E(m_2) = (g^{r_1}, g^{m_1} \cdot y^{r_1}) \times (g^{r_2}, g^{m_2} \cdot y^{r_2}) = (g^{r_1+r_2}, g^{m_1+m_2} \cdot y^{r_1+r_2}) = E(m_1 + m_2)$.

Distributed Encryption: works as follows: Let G denote a finite cyclic group of prime order q in which the decision Diffie-Hellman problem is intractable. Let g be a generator in G. There are n users, all of whom agree on (G, g). We assume there are n different users $u_1, u_2, \cdots, u_n$. Each user u_i chooses a secret value $x_i \in_R \mathbb{Z}_q$, and computes a public value g^{x_i}, where $1 \leq i \leq n$. Each u_i computes a y_i as below:

$$y_i = \frac{\prod_{j=1}^{i-1} g^{x_j}}{\prod_{j=i+1}^{n} g^{x_j}} \tag{1}$$

which is publicly computable since the computation uses all public values g^{x_i}.

We assume the message for each u_i is m_i, and the encrypted message $E(m_i, y_i, x_i)$ is $(y_i)^{x_i} \cdot g^{m_i}$, where $E(m_i, y_i, x_i)$ denotes the message m_i encrypted using y_i and x_i.

This cryptosystem also provides for **homomorphism addition**. Therefore, anyone can compute the sum of all ciphertexts simple by multiplying all encrypted messages:

$$\prod_{i=1}^{n} E(m_i, y_i, x_i) = \prod_{i=1}^{n} (y_i)^{x_i} g^{m_i} = \prod_{i=1}^{n} (y_i)^{x_i} \times \prod_{i=1}^{n} g^{m_i} = \prod_{i=1}^{n} g^{m_i} = g^{\sum_{i=1}^{n} m_i}$$

where $\prod_{i=1}^{n} (y_i)^{x_i} = 1$, according to [8].

Zero knowledge Proof: Our zero knowledge proof protocol is based on [15]. Given a cyclic group $G =< g >=< h >$ and public knowledge $A = g^x$ and $B = h^x$, the prover wants to convince verifier(s) A and B have the same exponentiation, but the verifier(s) cannot learn the value of x.

Prover: choose $t \in \mathbb{Z}_q$, computes $T_1 = g^t$, $T_2 = h^t$, $c = Hash(T_1||T_2)$, $s = x \cdot c + t$ sends T_1, T_2, s to **Verifier**

Verifier: computes $c = Hash(T_1||T_2)$, verifies if $g^s = A^c \cdot T_1$ and if $h^s = B^c \cdot T_2$

If both verifications are passed, the verifier believes the prover knows x, but cannot determine the value of x.

4 Our Proposed Voting System

In this section we present our proposed decentralized, self-tallying, ranked choice, smart contract-based voting system. The basic idea is as follows: The election administrator deploys a voting contract by confirming public parameters (such as the public key of the election). Each voter can then submit a vote via the voting contract, with each vote constituting a transaction of the blockchain system. In case of the vote not being verified as valid by the checks performed in the smart contract, the transaction reverts. After being mined by the blockchain's consensus algorithm, the vote is considered final. Figure 2 presents the stages of our proposed election.

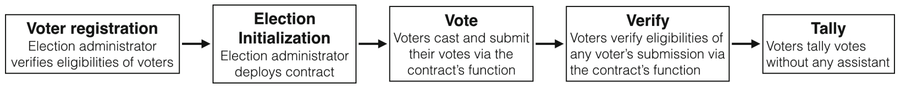

Voter registration
Election administrator verifies eligibilities of voters
Election Initialization
Election administrator deploys contract
Vote
Voters cast and submit their votes via the contract's function
Verify
Voters verify eligibilities of any voter's submission via the contract's function
Tally
Voters tally votes without any assistant

Fig. 2. The five steps of our proposed decentralized online voting system.

The involved participants in our proposed system are:

Election Administrator: An election administrator is required to set the election's parameters, begin the registration stage and add voters to the list of eligible voters. The administrator should also generate a key pair (public key and private key) of the election and contribute the public key to the blockchain. Furthermore, the administrator is responsible for voter registration, generating a candidate list, setting rules of the election, and deploying the voting contract, which cannot be changed once the election started.

Candidates: A list of candidates is generated by the election administrator. Each candidate is a contestant in the election and will receive points from different voters.

Voters: Each voter has a private key and public key. The public key is added to the blockchain after the eligibility of the voter is verified by the election administrator. The voter can submit his/her cast vote via the function provided by the smart contract.

Blockchain Database: A distributed and append-only database. All submissions will be added to the latest block of the chain once they are verified.

Table 1 provides the notations used to explain our protocols.

Table 1. Notations that used in the rest of the paper

n_c:	number of Candidates
n_v:	number of Voters
v_i:	i-th voter; $i \in [1, n_v]$
c_j:	j-th candidate; $j \in [1, n_c]$
$x_{v_i}^{c_j}$:	secret key of i-th voter that is used to vote for c_j candidate; $i \in [1, n_v], j \in [1, n_c]$
pk:	public key of election administrator
sk:	secret key of election administrator
P:	the pre-defined point, where the sum of all assigned points must equal to P
$p_{v_i}^{c_j}$	a point that is assigned by v_i to c_j, $i \in [1, n_v]$, $j \in [1, n_c]$
p^{c_j}	total received point of c_j, $j \in [1, n_c]$
ZPK{...}:	proof of zero knowledge

4.1 Initialization and Voter Registration

Before an election can start the cyclic group (G, p, g) is defined. The election administrator generates an ElGamal key pair (public key pk and secret key sk), and pk is added to the blockchain database, which can be accessed by all voters.

The only rule for defining the election parameters is that the sum of all assigned points must be a fixed number (which we treat it as P), the election administrator defines a list of the candidates and the value of P before the election starts.

In order to register, each voter must select n_c secret keys $x_{v_i}^{c_j} \in Z_p$ and compute the n_c corresponding public keys $g^{x_{v_i}^{c_j}} \pmod{p}$. The voter must register his real-world identification and his/her n_c pubilc keys to the election administrator. Once the eligibility is verified, the voter will be added to the list of eligible voters, and all his/her $g^{x_{v_i}^{c_j}}$ will be added to the blockchain. Once all eligible voters are registered for the election (or the deadline of registration has passed), the election administrator deploys the voting contract.

4.2 Voting Process

The proposed voting system allows voters to assign different scores to different candidates according to their personal preferences. There are three phases in the voting stage: pre-computing, vote casting and proof generation.

We do not remove the connection between the identity of voters and their votes, meaning everyone can see that a voter submitted his/her vote. However, the content of votes is encrypted, meaning no-one is able to reveal the content of any individual vote.

Pre-computing: We assume there are n_v registered voters, and all $g^{x_{v_i}^{c_j}}$ are viewable in the blockchain database. Thus, the pre-computing values $y_{v_i}^{c_j}$ of voters can be computed by using all other $g^{x_{v_i}^{c_j}}$ via Eq. 1. At the end, each v_i has n_c pre-computed values as $y_{v_i}^{c_1}, y_{v_i}^{c_2}, \cdots, y_{v_i}^{c_{n_c}}$, and each value can only be used to vote for the particular c_j.

Vote Casting: Each v_i is able to assign any integer point (from 0 to P) to different candidates, but the sum of all assigned points must equal to P (see Fig. 1), which is the rule each voter has to follow. Because each vote consists of multiple assigned points (according to the number of candidates), those points are treated as private and confidential to the voters. Thus, those scores must be encrypted before submission. In our case, we use $p_{v_i}^{c_j}$ to denote a score that is assigned by voter v_i to candidate c_j, which will be encrypted twice: ElGamal encryption and distributed encryption.

ElGamal Encryption. Each assigned point is encrypted using the public key pk of the election administrator. For example

$$E(p_{v_i}^{c_j}, pk) = g^r,\ g^{(p_{v_i}^{c_j})} \cdot pk^r$$

meaning the score $p_{c_j}^{v_i}$ is encrypted using pk according to the ElGamal encryption.

Distributed Encryption: Once the point is encrypted by ElGamal encryption, the first part (g^r) of the encrypted value will be "encrypted" again by using the private voting key (x_{v_i}) of the voter v_i, such as

$$g^r \rightarrow (y_{v_i}^{c_j})^{x_{v_i}^{c_j}} \cdot g^r$$

where $y_{v_i}^{c_j}$ is computed during the pre-computing phase and publicly accessible.

To summarise, we developed the encryption algorithm based on both the ElGamal encryption and group-based encryption, meaning each assigned point will be encrypted as per Eq. 2

$$E(p_{v_i}^{c_j}, pk, y_{v_i}^{c_j}, x_{v_i}^{c_j}) = (y_{v_i}^{c_j})^{x_{v_i}^{c_j}} g^r,\ g^{(p_{v_i}^{c_j})} \cdot pk^r \tag{2}$$

where $p_{v_i}^{c_j}$ is encrypted by using pk (public key of the election), $y_{v_i}^{c_j}$ (pre-computed value that is used by v_i to vote c_j) and $x_{v_i}^{c_j}$ (the particular private key of v_i to vote for c_j). Thus, a cast Vote_{v_i} (with n_c candidates) can be presented as:

$$\text{Vote}_{v_i} = \begin{bmatrix} E(p_{v_i}^{c_1}, pk, y_{v_i}^{c_1}, x_{v_i}^{c_1}) \\ \vdots \\ E(p_{v_i}^{c_{n_c}}, pk, y_{v_i}^{c_{n_c}}, x_{v_i}^{c_{n_c}}) \end{bmatrix}$$

Proof Generation: In order to allow anyone to verify the eligibility of each vote without decrypting the cipher text and revealing the content, each voter is required to generate several proofs for his/her vote before submission (ZKP denotes zero knowledge proof):

- $\text{ZKP}(x_{v_i}^{c_j})$: to prove each encrypted point for the candidate c_j is computed correctly using the voter's private key $x_{v_i}^{c_j}$.
- $\text{ZKP}(P)$: to prove that the sum of all encrypted points is equal to P.

The voter v_i has to generate $\text{ZKP}(x_{v_i}^{c_j})$ for each encrypted point $E(p_{v_i}^{c_j}, pk, y_{v_i}^{c_j}, x_{v_i}^{c_j})$, and $\text{ZKP}(P)$ for the Vote_{v_i}. The summarised processing procedure of the voting stage is shown in Algorithm 1.

Remark 1. The computation details about how to generate the $\text{ZKP}(x_{v_i}^{c_j})$ and $\text{ZKP}(P)$ can be found in **Appendix**.

4.3 Vote Verification Stage

In order to prevent multiple counting of any individual vote into the final result, vote verification is required as follows:

Verify Each Encrypted Point: In order to prevent having any error during tallying all submissions, each encrypted point $E(p_{v_i}^{c_j}, pk, y_{v_i}^{c_j}, x_{v_i}^{c_j})$ has to be confirmed as to have been computed with the correct parameters. The verification can be done by using the corresponding proofs $\text{ZKP}(x_{v_i}^{c_j})$ that are generated during vote casting.

Verify Sum of All Encrypted Points: According to the rules of the election, each voter cannot assign more than the pre-defined total available point P in his/her cast vote. Using homomorphic addition, anyone is able to compute the sum (encrypted) of all encrypted points and verify the value by using the corresponding proof $\text{ZKP}(P)$ that are generated by the voter.

The processing procedure of the verification is shown as Algorithm 2 (The purpose of function **verifyAddedVote** is similar, but the input parameters differ).

Algorithm 1. function **submitVote**

```
Input  : pre-defined point: $P$, public key $pk$,
         all secret keys of $v_i$: $x_{v_i}^{c_1}, \cdots, x_{v_i}^{c_{n_c}}$
         voting public keys of all voters $g^{x_{v_1}^{c_1}}, \cdots, g^{x_{v_{n_v}}^{n_c}}$
Output: $\text{Vote}_{v_i}$
1  computes $y_{v_i}^{c_j}$, $j \in [1, n_c]$.                                  ▷ refer to Eq. 1
2  set $\text{Vote}_{v_i} = []$
3  for $j \leftarrow 1$ to $n_c$ do
4      $E(p_{v_i}^{c_j}, pk, y_{v_i}^{c_j}, x_{v_i}^{c_j}) = \left((y_{v_i}^{c_j})^{x_{v_i}^{c_j}} \cdot g^r, g^{p_{v_i}^{c_j}} \cdot pk^r\right)$        ▷ refer to Eq. 2
5      $\text{ZKP}(x_{v_i}^{c_j})$: $\{K_1, K_2, Z_1, Z_2\}$                      ▷ Remark 1
6      $\text{Vote}_{v_i} = \text{Vote}_{v_i} \cup [E(p_{v_i}^{c_j}, pk, y_{v_i}^{c_j}, x_{v_i}^{c_j}), \text{ZKP}(x_{v_i}^{c_j})]$
7  end
8  $\text{ZKP}(P)$: $\{T_1, T_2, s, z\}$                                          ▷ Remark 1
9  $\text{Vote}_{v_i} = \text{Vote}_{v_i} \cup [\text{ZKP}(P)]$
10 $\text{Signature}_{v_i} = \text{Sign}(\text{Vote}_{v_i})$
11 if verifyPendingVote($\text{Vote}_{v_i}$) == False then
12     return False
13 end
14 return $\text{Vote}_{v_i} = \begin{bmatrix} E(p_{v_i}^{c_1}, pk, y_{v_i}^{c_1}, x_{v_i}^{c_1}), \text{ZKP}(x_{v_i}^{c_1}) \\ \vdots \\ E(p_{v_i}^{c_{n_c}}, pk, y_{v_i}^{c_{n_c}}, x_{v_i}^{c_{n_c}}), \text{ZKP}(x_{v_i}^{c_{n_c}}) \\ \text{ZKP}(P) \end{bmatrix}, \text{Signature}_{v_i}$
```

Remark 2. The computation details about how to verify the $\text{ZKP}(x_{v_i}^{c_j})$ and $\text{ZKP}(P)$ can be found in **Appendix**.

4.4 Votes Tally Stage

Once all voters have submitted their Vote_{v_i} and the deadline of submission has passed, the election administrator must do the following: (1) compute the tallying result (via homomorphic addition), (2) compute their partially decrypted value and proof; (3) send partially decrypted values (including proofs) to the blockchain.

Each point is encrypted using our developed encryption algorithm (Eq. 2), in which the cipher texts can be computed by homomorphic addition. In this case, we can simply multiply all Vote_{v_i} in the blockchain database as shown below, where we assume there are n_v voters and n_c candidates, and all Vote_{v_i} have been verified as valid.

$$\prod_{i=1}^{n_v} \text{Vote}_{v_i} = \begin{bmatrix} \prod_{i=1}^{n_v} E(p_{v_i}^{c_1} \cdots) \\ \vdots \\ \prod_{i=1}^{n_v} E(p_{v_i}^{c_{n_c}} \cdots) \end{bmatrix} = \begin{bmatrix} \prod_{i=1}^{n_v} (y_{v_i}^{c_1})^{x_{v_i}^{c_1}} g^{r_1},\ g^{\sum_{i=1}^{n_v} p_{v_i}^{c_1}} pk^{r_1} \\ \vdots \\ \prod_{i=1}^{n_v} (y_{v_i}^{c_{n_c}})^{x_{v_i}^{c_{n_c}}} g^{r_{n_c}},\ g^{\sum_{i=1}^{n_v} p_{v_i}^{c_{n_c}}} pk^{r_{n_c}} \end{bmatrix} \tag{3}$$

Algorithm 2. function **verifyPendingVote**

```
Input  : $\text{Vote}_{v_i}, g$, all $g^{x_{v_i}^{c_j}}$, all $y_{v_i}^{c_j}$
Output: Valid or Invalid
1  for $j \leftarrow 1$ to $n_c$ do
2      sum $*= E(p_{v_i}^{c_j}, \cdots)$
3      //verify $E(p_{v_i}^{c_j}, \cdots)$ using corresonding ZKP$(x_{v_i}^{c_j})$      ▷ Remark 2
4      $E(p_{v_i}^{c_j}, \cdots) = (c_1, c_2)$      ▷ refer to Algorithm 1
5      ZKP$(x_{v_i}^{c_j}) = \{K_1, K_2, Z_1, Z_2\}$      ▷ refer to Algorithm 1
6      compute $c = Hash(K_1 \| K_2)$
7      if $(y_{v_i}^{c_j})^{Z_1} g^{Z_2} \neq K_1 \times (c_1)^c \; OR \; g^{Z_1} \neq K_2 \times (g^{x_{v_i}^{c_j}})^c$ then
8          return False
9      end
10 end
11 //verify sum using corresonding ZKP$(P)$      ▷ Remark 2
12 assume sum $= (c_1, c_2)$
13 ZKP$(P) = \{T_1, T_2, s, z\}$      ▷ refer to Algorithm 1
14 compute $c = Hash(T_1 || T_2)$
15 if $(y_{v_i}^{c_j})^s \neq (\frac{c_1}{z})^c \cdot T_1 \; OR \; pk^s \neq (\frac{c_2}{g^P})^c \cdot T_2$ then
16     return False
17 end
18 return True
```

Due to $\prod_{i=1}^{n_v} (y_{v_i}^{c_j})^{x_{v_i}^{c_j}} = 1$ (refer to Sect. 3), $\prod_{i=1}^{n_v} \text{Vote}_{v_i}$ can be treated as n_c ciphertexts by ElGamal encryption, such as $E(\sum_{i=1}^{n_v} p_{v_i}^{c_j}), j \in [1, n_c]$.

The election administrator then has to compute partially decrypted values, such as $(g^{r_1})^{sk}, \cdots, (g^{r_{n_c}})^{sk}$. He/she must also generate the corresponding proof for each partially decrypted value to prove that each value is computed correctly using the secret key sk. Finally, the election administrator broadcasts the partially decrypted values (including the corresponding proofs ZKP(sk)) to the blockchain. The winner of the election can be computed by any voter with Algorithm 3.

Remark 3. The verification of each partial decrypted value can be treated as verifying if $(g^{r_j})^{sk}$ has the same exponentiation as pk, where $pk = g^{sk}$. The procedure is same as the example in Sect. 3.

Because the tallying algorithm is a function of the voting contract, the tallying result can be computed by any voter individually, without any key or decryption function.

5 Security Analysis

This section is devoted to a theoretical security analysis of our system. Noted that none of the previous related papers provided a formal security model, including only a description and an informal security discussion of their systems. Our

Algorithm 3. function **tallyVotes**

```
Input  : all valid votes $\text{Vote}_{v_1}, \cdots, \text{Vote}_{v_{n_v}}$ in blockchain
         all paritial decryption values $(g^{r_1})^{sk}, \cdots, (g^{r_{n_c}})^{sk}$ by election
administrator
Output: $p^{c_1}, \cdots, p^{c_{n_c}}$
1  compute $\prod_{i=1}^{n_v} \text{Vote}_{v_i}$                          ▷ refer to Eq. 3
2  //verify each partial decryption value using corresponding ZKP($sk$)
3  for $j \leftarrow 1$ to $n_c$ do
4      verify $(g^{r_j})^{sk}$ using corresponding ZKP($sk$)               ▷ Remark 3
5  end
6  //reveal result for all candidate using partial decryption values
7  for $j \leftarrow 1$ to $n_c$ do
8      $p_{c_j} = \frac{\prod_{i=1}^{n_v} g^{p_{v_i}^{c_1}} pk^{r_j}}{(g^{r_j})^{sk}} = \frac{g^{\sum_{i=1}^{n_v} p_{v_i}^{c_j}} (g^{sk})^{r_j}}{(g^{r_j})^{sk}} = g^{p_{v_1}^{c_j} + \cdots + p_{v_{n_v}}^{v_{c_j}}}$   ▷ refer to Sect. 3
9  end
10 return $p_{c_1}, p_{c_2}, \cdots, p_{c_{n_c}}$
```

analysis makes the following assumptions: (1) the election administrator and voters are always identifiable, as all blockchain transactions are signed with sender's private key. (2) Voters will never disclose their private voting keys $x_{v_i}^{c_j}$; (3) the blockchain database is secure and insert-only; (4) Our system relies on several cryptographic protocols, which are presented in Sect. 3 and have reliable published proofs of their security.

Theorem 1. *If the digital signature algorithm (such as DSA) is non-falsifiable, no one is able to submit a ballot by impersonating another voter.*

Proof. In order to prevent adversaries from casting ballots by impersonating authenticated voters, we require each voter to submit with his/her digital signature algorithm. In our proposed system, only eligible voters are added to the voters list by the election administrator once their identities have been verified. The signing_verify key of each verified voter is stored on the blockchain, and the voter is responsible for keeping their signing key secret. Once the election starts, each authorized voter signs their votes by using his/her signing key and submits the vote along with their signature. The smart contract is able to verify if each submission by verifying the digital signature using the corresponding signing_verify key.

Theorem 2. *Only one submission from each voter is accepted as valid.*

Proof. In our proposed system, only the content of a cast vote is encrypted, the identification of the voter (and the digital signature) is in plaintext and can be viewed by everyone. Thus, multiple-voting detection is achieved by our system, as it can always detect whether a voter has previously submitted a vote. Furthermore, depending on the requirements of the particular scenario, our system can accept one submission of each voter or accept multiple submissions for each voter and use the last vote as valid.

Theorem 3. *If the underlying cryptographic systems are semantically secure, then the votes' contents will never be revealed to anyone (including the election administrator).*

Proof. Every vote is encrypted twice before submission. We use the ElGamal cryptosystem and a distributed encryption algorithm, which inherits the homomorphic property from the standard ElGamal system. Both algorithms are semantically secure.

All the submitted votes remain in encrypted form as cipher texts all the time. The homomorphic property makes it possible to add all encrypted votes without decrypting them. Furthermore, there is no relationship between the cipher texts and the corresponding plaintexts since the cryptosystem employed is probabilistic. It applies random numbers, so that the cipher text can take on different values even when the encryption is computed from the same input. Finally, due to each value being encrypted by both the public key of the election administrator and the secret voting key of the voter, the decryption must be done via collaboration of the election administrator and the voter. This means that, if the voter kept his/her secret voting keys as secret all the time (that is also one of our assumptions), even the election administrator cannot reveal anything.

Theorem 4. *Integrity of all cast votes are secured after submission.*

Proof. Firstly, we require voters to sign their cast votes by using their signing keys (refer to Algorithm 1), and we assume voters do not share their signing keys, to ensure that nobody can modify the content of a submission and fake the voter's signature. Secondly, all cast votes will be verified being before being added to the blockchain. Third, all verified votes will be added to the blockchain, being logged in an immutable ledger. Thus, the integrity of all submitted votes is treated as secure.

Theorem 5. *Invalid votes can be detected by any individual voter.*

Proof. Each cast vote is added to the blockchain database with corresponding proofs, generated by using Zero Knowledge Proof. The verification algorithm is public to all voters, which means the voters are able to verify any vote without any assistant.

Theorem 6. *The self-tallying algorithm is proposed public accessible and anyone can use it to tally votes without assistant.*

Proof. Once all votes are verified and added to the blockchain, we require the election administrator to compute the partially decrypted value in order to allow voters to compute the tallied result by themselves. In the meantime, the election administrator must generate corresponding proofs to convince all voters that all partially decrypted values are computed correctly.

Theorem 7. *Voters are able to verify everything of the election.*

Proof. In our system, all content (encrypted votes, proofs and signature) for each submission is broadcasted and added to the blockchain database, where they can be accessed by anyone. We assume that the blockchain database is secure, and it is "append-only". The voters can do the following without any assistant: (1) Voters can verify the blockchain transactions themselves. (2) Voters can verify the integrity of each submission by using the corresponding signing_verify keys from all voters. (3) Voters can verify each partially decrypted value (computed by election administrator) is computed correctly. (4) Voters can self-tally all votes and compute the final result of the election.

6 Performance Analysis

This section discusses the performance of our proposed voting system. The analysis is based on the computation time of each processing step, separated into 3 phases, vote casting performance, votes verification performance and votes tallying performance. In our proposed protocol, each vote is encrypted twice using different keys (common key of election administrator and secret key of the voter, refer to Algorithm 1). All tests were performed using a 512-bit key (p is 512-bit), which provides a higher security level than one-time encryption using a 1024-bit key.

We tested our proposed protocol using a high performance implementation of libgmp via the gmpy2 python module (https://gmpy2.readthedocs.io/en/latest/), on a laptop with the following specifications: 2.8 GHz quad-core Intel Core i7 with 6 MB shared L3 cache and with 16 GB of 1600 MHz DDR3L on-board memory.

We use t to denote the computation time of one exponentiation, where $t = 0.09$ ms. ElGamal encryption requires two exponentiations, and ElGamal decryption requires one exponentiation, where the division can be avoided by using an alternative method (https://wikipedia.org/wiki/ElGamal_encryption). Thus, we use t_E and t_D to denote the computation time of encryption and decryption, respectively, where $t_E = 2t$ and $t_D = t$, approximately. Pre-computed values of distributed encryption (refer to Eq. 1) require one exponentiation (the inverse power computation), and encryption also has cost of one exponentiation.

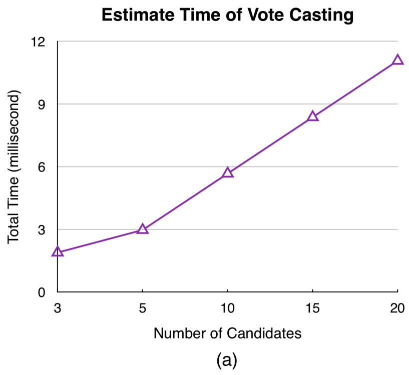
6.1 Vote Casting Performance

The performance can be analysed for the following aspects:

Total Computation Time: According to the Algorithm 1, we use T_{voter} to denote the total time spent before submission (including the proof generation time), where

$$T_{voter} = (t_E \times n_c + t * n_c) + (3 * n_c * t) + (3 * t) = (6n_c + 3)t$$

In this experiment, we tested T_voter in five rounds, varying the number of candidates ($n_c = 3, 5, 10, 15, 20$). The result is shown in (a) Fig. 3. From the results in (a) Fig. 3, we can see the time cost for casting a vote is less than 12 ms even if there are 20 candidates to be ranked.



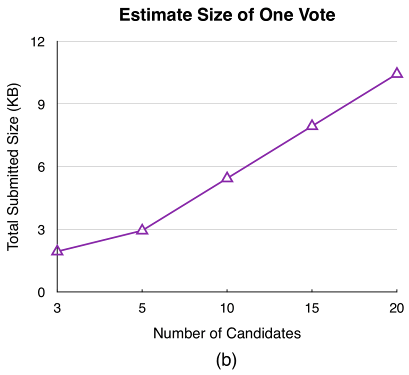


Fig. 3. Performance of voter side when the number of candidates is 3, 5, 10, 15, 20: (a) Time spent encrypting a cast vote, including generation time of all proofs (b) The size of a submission, includes all encrypted values and all proofs.

Total Submission Size: We assume the size of digital signature is 1024-bit (refer to Algorithm 1), and we use S_{vote} to denote the total submission size (bits) for a voter,

$$S_{vote} = (1024 \times n_c) + (2048 \times n_c) + (2048 + 512 + 2 * n_c * 512) + (1024) = 4096 * n_c + 3584$$

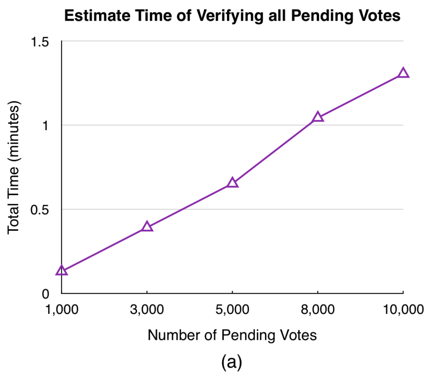
The test result is shown in (b) Fig. 3 based on different numbers of candidates ($n_c = 3, 5, 10, 15, 20$). From the result of (b) Fig. 3, we found the submission size of one vote is less than 11 KB even for a 20-candidate ballot.

6.2 Votes Verification Performance

We have also evaluated the performance of the verification time for submissions a member of the public or an independent observer might with to verify. Due to the verification of each voter's identification being equivalent to verifying the digital signature of each submission, this is not computationally expensive. Thus, we concentrated on the performance of Algorithm 2. We use T_{verify} to denote the total time spent verifying votes and n to denote the total number of votes being verified, which can be presented as follows:

$$T_{verify} = \big((5t \times n_c) + (n_c + 1 + 2 * n_c)t + (6t)\big) \times n = (8 * n_c + 7)t$$

We tested T_{verify} in five rounds, varying the numbers of votes verified (n = 1000, 3000, 5000, 8000, 10000). In this experiment, we assume the number of candidates is 10 (n_c = 10), and the result is shown in Fig. 4(a). From the results in Fig. 4, we found the time spent verifying 10, 000 ballots costs less than 1.5 min.



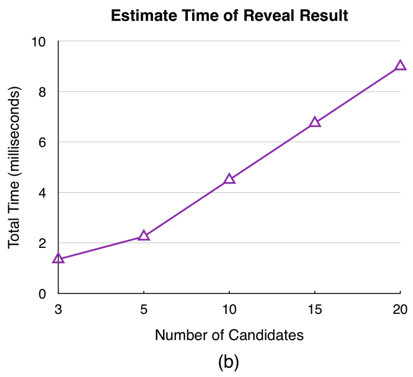


Fig. 4. (a) Estimate time spent of verifying 1000, 3000, 5000, 8000, 10000 votes. (b) Estimate time spent of tallying all votes, including verifying all partially decrypted values.

6.3 Votes Tallying Performance

Our proposed system allows voters to self-tally all submitted votes by using all partially decrypted values from the election administrators. However, before tallying starts, each partially decrypted value must be verified using the corresponding proofs (refer to Sect. 4.4). T_{reveal} is used to denote the total time spent verifying all partially decrypted values, tally all votes, reveal the result (refer to Algorithm 3), which is presented as:

$$T_{reveal} = (4t * n_c) + (n_c * t) = 5 * n_c * t$$

Again, we tested T_{tally} in five rounds varying the number of candidates ($n_c = 3, 5, 10, 15, 20$). The result of this experiment is shown in Fig. 4(b). We found the time spent tallying all votes (including verifying all partial decryption proofs) costs less than 10 ms using the same test machine.

7 Conclusion and Future Work

We have proposed a secure decentralized online voting system using cryptography and a smart contract, which allows the voters to cast their ballots by assigning arbitrary numbers of points to different candidates. This means that the voters can assign equal points to different candidates, or they can assign different points to different candidates. Each cast vote in our system is encrypted before submission and remains encrypted at all times. The additive homomorphic property of the ElGamal cryptosystem enables effective processing of the cipher texts during these procedures. Furthermore, the eligibility of voters and their submissions can be verified by anyone without revealing the contents of the ballots. The security and performance analysis confirm the feasibility of our proposed cryptographic voting contract.

There is a limitation in that our system may suffer abortive issue [10,14]. We have to assume that all registered voters submit their valid votes since otherwise, any voter can abort the tally without submitting his/her vote. In future work, we will address this issue and consider further generalizations. Furthermore, we will migrate the whole system to an Ethereum network and perform trials of the protocol at a larger scale.

Appendix

In order to protect the privacy of the voters, each assigned point is encrypted (refer to Eq. 2) before submission. However, voters have to generate corresponding proofs to prove their votes are cast correctly by observing the following: 1. each encrypted score is computed correctly using the voter's private key $x_{v_i}^{c_j}$; 2. sum of all encrypted points are equaled to pre-defined total available point P.

Each Encrypted Value is Computed Correctly: We present the proofs generation and verification for an encrypted score in Vote$_{v_1}$, where we assume v_1 assigned 5 points to c_1, and the encrypted value should be $E(5, pk, y_{v_1}^{c_1}, x_{v_1}) = (c_1, c_2)$, where $c_1 = (y_{v_1}^{c_1})^{x_{v_1}^{c_1}} \cdot g^r$, $c_2 = g^5 \cdot pk^r$:

Prover: generates a random number $k_1, k_2 \in \mathbb{Z}_q$, computes $K_1 = (y_{v_1}^{c_1})^{k_1} \cdot g^{k_2}$, computes $K_2 = g^{k_1}$, $c = Hash(K_1 \| K_2)$, $Z_1 = x_{v_1}^{c_1} c + k_1$, $Z_2 = rc + k_2$. And ZKP$(x_{v_i}^{c_1}) = \{K_1, K_2, Z_1, Z_2\}$

Verifier: compute $c = Hash(K_1 \| K_2)$, verify if $(y_{v_1}^{c_1})^{Z_1} g^{Z_2} = K_1 \times (c_1)^c$, verify if $g^{Z_1} = K_2 \times (g^{x_{v_i}^{c_1}})^c$, where $y_{v_i}^{c_1}$ and $g^{x_{v_i}^{c_1}}$ are public values, and the verifier(s) will never know the value is encrypted from 5.

Sum of All Encrypted Values is Equavelent to Encrypted from P: We present the proofs generation and verification for the sum of all encrypted points in Vote$_{v_1}$, where we assume $P = 10$, and there are 3 candidates c_1, c_2 and c_3. Voter v_1 cast a vote as:

$$E(5, pk, y_{v_1}^{c_1}, x_{v_1}^{c_1}) = (y_{v_1}^{c_1})^{x_{v_1}^{c_1}} \cdot g^{r_1}, g^5 \cdot pk^{r_1}$$
$$E(2, pk, y_{v_1}^{c_2}, x_{v_1}^{c_2}) = (y_{v_1}^{c_2})^{x_{v_1}^{c_2}} \cdot g^{r_2}, g^2 \cdot pk^{r_2}$$
$$E(3, pk, y_{v_1}^{c_3}, x_{v_1}^{c_3}) = (y_{v_1}^{c_3})^{x_{v_1}^{c_3}} \cdot g^{r_3}, g^3 \cdot pk^{r_3}$$

Prover: multiply them as

$$\begin{aligned} & E(5, pk, y_{v_1}^{c_1}, x_{v_1}^{c_1}) \times E(2, pk, y_{v_1}^{c_2}, x_{v_1}^{c_2}) \times E(3, pk, y_{v_1}^{c_3}, x_{v_1}^{c_3}) \\ =& (y_{v_1}^{c_1})^{x_{v_1}^{c_1}} \cdot (y_{v_1}^{c_2})^{x_{v_1}^{c_2}} \cdot (y_{v_1}^{c_3})^{x_{v_1}^{c_3}} \cdot g^{r_1+r_2+r_3}, g^{5+2+3} \cdot pk^{r_1+r_2+r_3} \\ =& (y_{v_1}^{c_1})^{x_{v_1}^{c_1}} \cdot (y_{v_1}^{c_2})^{x_{v_1}^{c_2}} \cdot (y_{v_1}^{c_3})^{x_{v_1}^{c_3}} \cdot g^{r_4}, g^{10} \cdot pk^{r_4} \end{aligned}$$

We use c_1 and c_2 to denote $(y_{v_1}^{c_1})^{x_{v_1}^{c_1}} \cdot (y_{v_1}^{c_2})^{x_{v_1}^{c_2}} \cdot (y_{v_1}^{c_3})^{x_{v_1}^{c_3}} \cdot g^{r_4}$ and $g^{10} \cdot pk^{r_4}$, respectively.

And then, compute $z = \prod_{j=1}^{n_c}(y_{v_1}^{c_j})^{x_{v_1}^{c_j}}$. In this case, $z = (y_{v_1}^{c_1})^{x_{v_1}^{c_1}} \cdot (y_{v_1}^{c_2})^{x_{v_1}^{c_2}} \cdot (y_{v_1}^{c_3})^{x_{v_1}^{c_3}}$. Selects random numbers $k_1, k_2, k_3 \in \mathbb{Z}_q$, computes $K_1 = (y_{v_1}^{c_1})^{k_1} \cdot (y_{v_1}^{c_2})^{k_2} \cdot (y_{v_1}^{c_3})^{k_3}$, $K_2 = g^{k_1}$, $K_3 = g^{k_2}$, $K_4 = g^{k_3}$, $c = Hash(K_1 \| K_2 \| K_3 \| K_4)$, $Z_1 = x_{v_1}^{c_1} c + k_1$, $Z_2 = x_{v_1}^{c_2} c + k_2$, $Z_3 = x_{v_1}^{c_3} c + k_3$. And ZKP$(z) = \{K_1, K_2, K_3, K_4, Z_1, Z_2, Z_3\}$.
Prove $\frac{c_1}{z}(= g^{r_4})$ and $\frac{c_2}{g^{10}}(= pk^{r_4})$ has the same exponentiation (refer to Sect. 3), select $t \in Z$, compute $T_1 = (g)^t$, compute $T_2 = pk^t$, compute $c = Hash(T_1 || T_2)$, compute $s = r_4 \cdot c + t$ (r_4 in this case). ZKP$(P) = \{T_1, T_2, s, z, \text{ZKP}(z)\}$.

Verifier: firstly verify z using ZKP(z), compute $c = Hash(K_1 \| K_2 \| K_3)$, verify if $(y_{v_1}^{c_1})^{Z_1}(y_{v_1}^{c_2})^{Z_2}(y_{v_1}^{c_3})^{Z_3} = K_1 \times (z)^c$, verify if $(y_{v_1}^{c_1})^{Z_1} = K_1 \times (g^{x_{v_1}^{c_1}})^c$, verify if $(y_{v_1}^{c_2})^{Z_2} = K_2 \times (g^{x_{v_1}^{c_3}})^c$, verify if $(y_{v_1}^{c_3})^{Z_3} = K_3 \times (g^{x_{v_1}^{c_3}})^c$.
Secondly verify P using T_1, T_2, s and z, multiply $E(5, \cdots)$, $E(2, \cdots)$ and $E(3, \cdots)$ as (c_1, c_2), compute $c = Hash(T_1 || T_2)$, verify if $g^s = (\frac{c_1}{z})^c \cdot T_1$, verify if $pk^s = (\frac{c_2}{g^{10}})^c \cdot T_2$. Same as to verify $\frac{c_1}{z}$ and $\frac{c_2}{g^{10}}$ has the same exponentiation r_4.

References

1. Adida, B.: Helios: web-based open-audit voting. In: USENIX Security Symposium, vol. 17, pp. 335–348 (2008)
2. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
3. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Advances in Cryptology, pp. 10–18 (1984)
4. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
5. Buterin, V.: A next-generation smart contract and decentralized application platform (2015). https://github.com/ethereum/wiki/wiki/White-Paper
6. Szabo, N.: Smart contracts: building blocks for digital markets. EXTROPY J. Transhumanist Thought **16** (1996)
7. Followmyvote.com. Introducing a secure and transparent online voting solution for the modern age: Follow My Vote (2016). https://followmyvote.com/
8. Hao, F., Ryan, P.Y., Zieliński, P.: Anonymous voting by two-round public discussion. IET Inf. Secur. **4**(2), 62–67 (2010)
9. Groth, J.: Efficient maximal privacy in boardroom voting and anonymous broadcast. In: International Conference on Financial Cryptography, pp. 90–104 (2004)
10. Kiayias, A., Yung, M.: Self-tallying elections and perfect ballot secrecy. In: International Workshop on Public Key Cryptography, pp. 141–158 (2002)
11. Li, M., Sun, X., Wang, H., Zhang, Y., Zhang, J.: Privacy-aware access control with trust management in web service. World Wide Web **14**(4), 407–430 (2011)
12. Kabir, M.E., Wang, H.: Conditional purpose based access control model for privacy protection. In: Proceedings of the Twentieth Australasian Conference on Australasian Database, vol. 92, pp. 135–142 (2009)
13. Kabir, M.E., Wang, H., Bertino, E.: A role-involved purpose-based access control model. Inf. Syst. Front. **14**(3), 809–822 (2012)
14. McCorry, P., Shahandashti, S.F., Hao, F.: A smart contract for boardroom voting with maximum voter privacy. In: International Conference on Financial Cryptography and Data Security, pp. 357–375 (2017)

15. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991)
16. Sun, X., Li, M., Wang, H.: A family of enhanced (L, α)-diversity models for privacy preserving data publishing. Future Gener. Comput. Syst. **27**(3), 348–356 (2011)
17. Sun, X., Wang, H.: Satisfying privacy requirements before data anonymization. Comput. J. **55**(4), 422–437 (2012)
18. Sun, X., Wang, H., Li, J., Truta, T.M.: Enhanced p-sensitive k-anonymity models for privacy preserving data publishing. Trans. Data Priv. **1**(2), 53–66 (2008)
19. Wang, H., Cao, J., Zhang, Y.: A flexible payment scheme and its role-based access control. IEEE Trans. Knowl. Data Eng. **17**(3), 425–436 (2005)
20. Wang, H., Zhang, Y., Cao, J.: Effective collaboration with information sharing in virtual universities. IEEE Trans. Knowl. Data Eng. **21**(6), 840–853 (2009)
21. Business Wire. Now you can vote online with a selfie. Business Wire (2016). http://www.businesswire.com/news/home/20161017005354/en/Vote-Online-Selfie
22. Yang, X., et al.: A verifiable ranked choice internet voting system. In: Bouguettaya, A., et al. (eds.) WISE 2017. LNCS, vol. 10570, pp. 490–501. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68786-5_39
23. Yang, X., Yi, X., Nepal, S., Kelarev, A., Han, F.: A secure verifiable ranked choice online voting system based on homomorphic encryption. IEEE Access (2018)