Enhanced Boardroom Voting with Block chain and Smart Contract

Huiqin Xie

School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

xiehuiqin@iie.ac.cn

Corresponding author

Keywords: Smart Contract, Ethereum, Blockchain, Self-tallying voting protocol, Decentralization

Abstract: We present two decentralized and self-tallying anonymous voting protocols with multiple candidates using Ethereum block chain, one of which requires to use multiple generators of an elliptic curve group, while the other one uses only a single generator. We prove that both protocols fulfill fairness, maximum ballot secrecy and dispute-freeness under the partial-collusion attack model. Their correct execution and security are guaranteed by the underlying consensus mechanism of Ethereum and thus no trusted authorities are required. We rigorously analyze the efficiency of the protocols from the perspectives of computational complexity and communication complexity. Also, we estimate the total payments of the administrator and the voters for the multiple-generator construction and single-generator construction respectively and make comparisons.

1. Introduction

The notion of block chain was first proposed in the Bitcoin cryptocurrency system [1]. Its initial motivation is to support Bitcoin system's financial transactions while removing the centralized role of banks, but now it has been further applied to many other industry fields, such as supply chain, legal and medical. The block chain can be viewed as an append-only ledger, maintained by an open-membership decentralized peer-to-peer network. The security of block chain is guaranteed by the consensus mechanism [2], which promotes agreement on the transaction history.

Since the emerge of the Bitcoin, other block chain-based systems have been created that extend the function of the block chain beyond being a cash system and allow to express other kinds of applications on the block chain as "smart contracts". One such system is Ethereum [3]. Ethereum block chain is a globally distributed computing platform which executes smart contracts and allows to build several of decentralized applications on it with its built-in economic functions. Ethereum's development plan is divided into four different stages, each of which will undergo major changes. The four stages are named Frontier, Homestead, Metropolis and Serenity. Current stage is the Metropolis.

Many applications of smart contracts have been proposed. Bonneau et al. embed the core data structures of CONIKS, a key transparency system for maintaining the directory of users' public keys, into a smart contract with minor modifications [4]. Users of the resulting system can rely on the Ethereum's network to audit data structures. Due to several drawbacks of the public key infrastructure (PKI) arising from its centralized design, Al-Bassam proposed an alternative PKI system with decentralized and transparent properties using a smart contract [5]. The contract-based PKI makes it easy to detect rogue certificates when they are published. To achieve verifiable cloud computing at a reasonable cost, Dong et al. applied smart contracts to the betrayal, tension and distrust between the clouds [6]. They demonstrated that, after introducing smart contracts, any rational cloud will not cheat or collude according to game theory. To defend against the dishonest behaviors of domain in using certificates during Transport Layer Security (TLS) handshake authentication, Xia et al. proposed a block chain-based system ETDA based on IKP and certificate transparency [7]. In the system, a smart contract is used to enforce the automatic punishment for

domain's misbehavior and avoids domains sending invalid certificates to users. McCorry et al. implemented a decentralized internet voting protocol that is self-tallying and has maximum voter privacy using smart contracts over Ethereum [8]. Their voting protocol confines to the case where there is only two options to vote, such as yes/ No.

Besides the applications of Ethereum, some works for improving Ethereum smart contract have also been done. Zhang et al. presented an authenticated data feed system named Town Crier that serves as a bridge between Ethereum smart contracts and existing websites [9]. Amani et al. extended an existing Ethereum Virtual Machine (EVM) formalization in Isabelle/HOL with a program logic at the bytecode level, which helps to control the complexity and cost of formal verification of smart contracts on EVM [10].

The decentralization, auditability and transparency natures of Ethereum is especially useful for the implementation of decentralized anonymous voting. There have been a great number of works aimed at reducing the dependency of the security of voting protocols on trusted authorities. A standard approach that has been used in many election protocols is to distribute the trust among multiple third parties by a threshold scheme, such that the security does not depend on a single trusted authority [11, 12]. However, this still requires the trust for the third parties altogether. To solve this problem, Kiayias and Yung proposed a self-tallying voting protocol that provided maximum privacy of voters [13]. It assumes the existence of a bulletin board, which is a public broadcast channel with memory and used for all communications between the participants. However, the protocol requires heavy computational load for the voters. Afterwards, Groth studied the efficiency limits of the Kiayias-Yung protocol and proposed a new voting scheme with better computational complexity [14]. Unfortunately, Groth's protocol requires large numbers of rounds. Based on these studies, F. Hao et al. proposed a two-round self-tallying anonymous voting scheme with small computational load and maximum privacy of voters [15]. The protocol assumes the existence of a bulletin board and a public authenticated channel accessible for each voter. The decentralized internet voting protocol implemented by McCorry et al. using smart contracts is based on the two-candidate case of this protocol.

In this paper, we present a decentralized anonymous voting protocol that is self-tallying and support multiple-candidate case based on Ethereum smart contracts. In the protocol, the Ethereum block chain serves as the public bulletin board and the underlying peer-to-peer network is used as an authenticated channel. The consensus mechanism which guarantees the security of Ethereum enforces the correct implementation of the protocol. Specifically, our primary contributions are:

- We propose two ways to implement a decentralized and self-tallying anonymous voting protocol over Ethereum for multiple-candidate case, one of which uses multiple generators of an elliptic curve group while the other one uses only single generator. The execution and security rely on the underlying consensus mechanism of Ethereum and do not require any trusted authority.
- We prove that the protocol fulfills completeness, robustness, eligibility, unreusability, fairness, maximum ballot secrecy and dispute-freeness under the partial-collusion attack model.
- We rigorously analyze the efficiency of the protocol from the perspectives of computational complexity and communication complexity. Also, we estimate the total payments of the administrator and the voters for both the multiple-generator case and single-generator case and compare these two kinds of implementations.

2. Background

In this section, we briefly recall preliminary notions concerning Ethereum. From the perspective of computer science, Ethereum is a globally distributed computing platform which executes programs called smart contracts. It utilizes a block chain to store and synchronize the state of system, and a cryptocurrency named ether (or ETH) to measure and restrict the costs of execution resource.

There are two types of accounts in Ethereum. One is called externally owned account (EOA). An EOA is actually a public-private key pair and controlled by a person or an external server who has the private key. Another type of account is a smart contract account which has no private key and is

controlled by the logic if its code. The programs wrote on smart contracts are run on the emulated computer called Ethereum Virtual Machine (EVM). Both types of account can store ethers.

A smart contract will not be executed unless it is originally activated by a user-owned account. That is, smart contracts is allowed to call other contracts directly, but the origin of a series of calls must be an EOA. Externally owned accounts create and call smart contracts by sending transactions to relevant addresses, e.g. the zero address when creating a smart contract. The basic structure of transactions in Ethereum contains the following six fields:

-From: a digital signature of an originating EOA.

-To: destination Ethereum address and can be either an EOA or a contract address.

-Data: variable length binary data payload, generally contains the code of a contract to be created or execution instructions for called contract.

-Gas Price: the price of gas that the originating user is willing to pay using ether currency.

-Gas Limit: the maximum amount of gas the originating user is willing to pay for this transaction.

-Nonce: the number of transactions that have been sent from the originating address.

Valid transactions will eventually be included in the Ethereum block chain by the miners, causing a global state transition. The Ethereum block chain can be viewed as a transaction-based orderly state machine. If a smart contract is called by several transactions, then the final state of the contract is determined by the order in which transactions are stored in the block chain. The security of the block chain is guaranteed by the consensus mechanism called proof of work (POW). Proof of work is essentially a computationally difficult puzzle, and the miner who finds the solution is allowed to append a new block. The POW algorithm used by Ethereum is called Ethash and makes use of an evolution of the Dagger-Hashimoto algorithm. Solving Ethash requires to maintain and frequently access a large database, which makes it "ASIC resistant". That is, it is difficult to produce Application Specific Integrated Circuits (ASIC) mining equipment that solves Ehash much faster than a GPU. The miner who successfully appends a new block can get 5 ETH as a reward. Beside this, he also receives all the gas in fees cost by the transactions in the block he appends. The POW consensus mechanism has been proved to fulfill two security properties: persistence and liveness. Persistence states that once a transaction is contained in a block that has depth larger than some security parameter k in the block chain of an honest node, then it will be included in the same position in the block chain held by each honest player with overwhelming probability. Liveness ensures that all transactions originating from honest nodes will eventually are included in an honest node’s block chain with depth larger than k.

The storage and computational resources on Ethereum is measured and limited by gas. Each user must set the maximum amount of gas he is willing to pay for the transaction before sending it. If the gas is run out before all operations are completed, the execution will be halted and the transaction will be reverted. This mechanism helps to prevent accidental or malicious wastage of computational resource of the network. Each basic operation executed by a contract or a transaction has a fixed cost in gas. For examples, an addition of two numbers takes 3 gas, and sending a transaction requires 21,000 gas. If the execution of a transaction finishes successfully, the gas cost for the execution will be converted to ether according to the gas price specified by the sender and paid to the miner as a transaction fee:

$$Miner\ fee = gas\ cost \times gas\ price \tag{1}$$

The remaining gas will be refunded to the sender

The Ethereum provides a platform for building powerful decentralized applications. Its natures of high auditability, availability and transparency make it suitable for building decentralized voting protocols. The block chain can be used as a public bulletin board of the voting scheme, and its underlying peer-to-peer network can serve as an authenticated channel. The rules of the voting protocol are converted to the code logic of a smart contract, and voters cast ballots by interacting with the smart contract as shown in Fig 1. The correct implementation of the voting is enforced by

the consensus mechanism which guarantees the security of Ethereum.

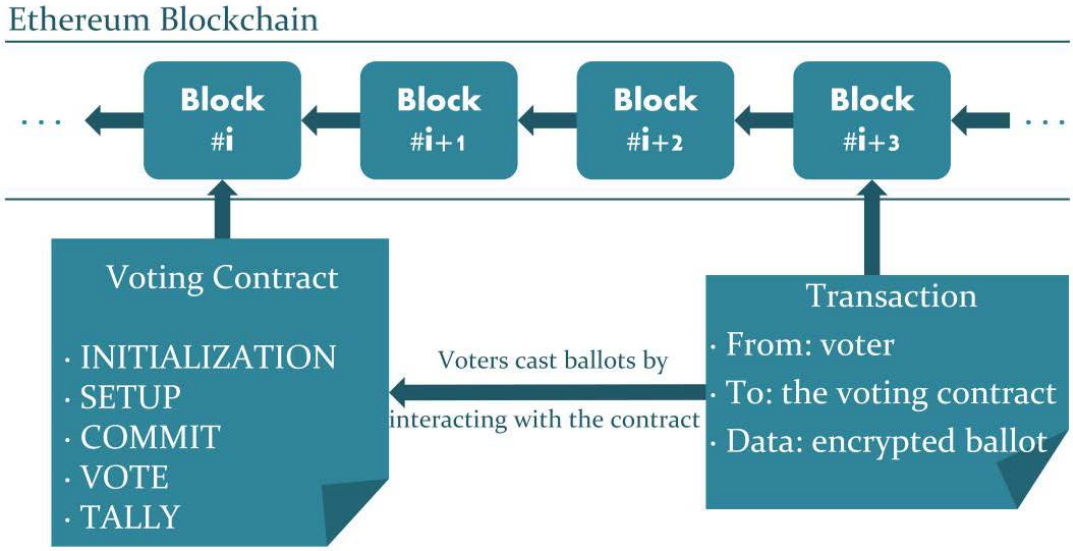

Figure 1. Schematic diagram of implementing a voting protocol on Ethereum.

3. Self-tallying anonymous voting protocol

In this section we present two multiple-candidate boardroom voting protocols before discussing how to implement them over Ethereum. Both voting protocols are based on the two-round anonymous election scheme proposed by Hao. F et al. [15]. The election scheme in [15] is self-tallying. That is, it does not require a trusted authority to tally the ballots. Instead, any voter or any third-party observer can execute the tally computation by themselves via an open procedure. This accords with the block chain's core design principle of decentralization. However, the self-tallying scheme has the fairness drawback that the last voter is able compute the tally ahead of others, which may lead to adaptive problems. The last voter can decide his own vote according to other voters' choices. To address this problem, we add an extra round before the formal vote where all voters are required to publish the hash of their encrypted vote so that any voter, including the last one, cannot change his vote even though he can compute the tally.

The protocols in this paper assume an authenticated public channel accessible for all participants. This is a basic requirement for election protocols and the underlying peer-to-peer network of the Blockchain can serve as such an authenticated channel. The anonymous election scheme proposed by Hao. F et al. is originally for the case where there is only two options, such as yes/ No. To extend the scheme to multiple-candidate case, they offer two different methods. One uses multiple independent generators of the finite field to represent different candidate. The other one uses only a single generator but the plaintext of ballots have larger exponents. Our two boardroom voting protocols also respectively apply these two methods. In the following we present the description of both protocols, and in the fifth section we will compare their efficiency after combining with smart contracts.

3.1 Voting Protocol Based on Multiple Generators

Suppose there are n voters and k candidates. The eligible voters are denoted by $P_1, P_2, \ldots, P_n$, respectively. Let E denote an elliptic curve over a finite field $\mathbb{F}_q$ where the Decisional Diffie-Hellman problem is intractable. Let $G, G_1, G_2, \ldots, G_k$ be k+1 independent generators of E. for an arbitrary point $Z \in E$, we use the notation $Log_G Z$ to denote the scalar $z \in \mathbb{Z}_q$ such that $Z = zG$. let H denote some publicly agreed collision-resistant hash function. Each voter P_i selects a value $x_i \in_R \mathbb{Z}_q$ at random as their private key. The voters execute the following three-round protocol:

Protocol 1.

Round 1. Each voter P_i broadcasts the voting key x_iG and a non-interactive zero-knowledge proof $NIZKP(x_i)$ to prove the knowledge of the scalar x_i on the bulletin board. The proof $NIZKP(x_i)$ is generated by Schnarr's signature [16] and Fait-Shamir heuristic [17], and we explain the detailed computations below. At the end of this round, every voter checks the validity of all zero-knowledge proofs and then computes a set of reconstructed keys:

$$Y_i = \sum_{j=1}^{i-1} x_j G - \sum_{j=i+1}^{n} x_j G, \quad i = 1, 2, \ldots n. \tag{2}$$

Suppose $Y_i = y_i G$ for some $y_i \in \mathbb{Z}_q$, then it holds that $\sum_i x_i y_i = 0$.

Round 2. Each voter P_i calculates the encrypted vote $x_i y_i G + Q_i$, where $Q_i \in \{G_1, G_2, \ldots, G_k\}$ and $Q_i = G_j$ if P_i chooses the jth candidate. Then he commits his ballot by publishing the hash of the encrypted vote $H(x_i y_i G + Q_i)$.

Round 3. Each voter P_i publishes $x_i y_i G + Q_i$ and a non-interactive zero-knowledge proof $NIZKP(Q_i)$ to prove that Q_i is one of $\{G_1, G_2, \ldots, G_k\}$ without leaking which one. This one-out-of-k zero-knowledge proof is generated by CDS technique [18] and Fait-Shamir heuristic, and we explain the detailed computations below. All zero-knowledge proofs and commitments need to be verified before tallying.

Once all votes have been cast, anyone can compute $\sum_i (x_i y_i G + Q_i)$. Since $\sum_i x_i y_i = 0$, it is equal to $\sum_i Q_i = c_1 G_1 + c_2 G_2 + \ldots + c_k G_k$, where $c_1, c_2, \ldots, c_k$ are the corresponding counts of the votes for k candidates. The tally $c_1, c_2, \ldots, c_k$ are all small numbers, so they can be calculated by exhaustive search. There are $C_{n+k-1}^{k-1} = O(n^{k-1})$ possible voting results. The exhaustive search is feasible when k is a small number.

In the above protocol, each voter needs to compute two non-interactive zero-knowledge proofs. In Round 1, each voter P_i computes the zero-knowledge proof $NIZKP(x_i)$ to prove his knowledge of the scalar x_i by Schnarr's signature and Fait-Shamir heuristic. Specifically, P_i chooses a random $v \in_R \mathbb{Z}_q$ and calculates the zero-knowledge proof $(vG, r = v - x_i z)$, where $z = H(G, vG, x_i G, i)$. To verify the proof, one only needs to check whether vG is equal to $rG + z x_i G$.

In Round 3, each voter P_i computes the zero-knowledge proof $NIZKP(Q_i)$ to prove that Q_i is one of $\{G_1, G_2, \ldots, G_k\}$ by CDS technique. Specifically, P_i first converts the encrypted vote to the form of the ElGamal encryption

$$(x_i G,\ x_i (y_i G) + Q_i). \tag{3}$$

This is exactly the Megamall encryption of Q_i with a public key $y_i G$ and randomness x_i. When given a general ElGamal encryption $(X, K) = (x_i G, x_i Y + M)$, the CDS protocol proves that M is one of $\{M_1, M_2, \ldots, M_k\}$ without leaking which through proving the following statement

$$Log_G X = Log_Y (K - M_1) \vee Log_G X = Log_Y (K - M_2) \vee \ldots \vee Log_G X = Log_Y (K - M_k). \tag{4}$$

Fig 2. Represents a three-round interactive protocol using CDS technique to prove the above statement with $Y = y_i G$ and $M_1 = G_1, M_2 = G_2, \ldots, M_k = G_k$. By Fait-Shamir heuristic we can convert it into a non-interactive proof. Specifically, let the challenge c in the protocol be the hash value

$$H(i, X, Y, A_1, \ldots, A_k, B_1, \ldots, B_k), \tag{5}$$

Instead of randomly chosen by the verifier, then the non-interactive proof is

$$(A_1, A_2, \ldots, A_k, B_1, B_2, \ldots, B_k, d_1, d_2, \ldots, d_k, r_1, r_2, \ldots, r_k), \tag{6}$$

Where $d_1, d_2, \ldots, d_k, r_1, r_2, \ldots, r_k$ are generated as the protocol in Fig 2 while using the challenge c defined in Eq. (5). For further details about one-out-of-k proof, one can refer to [18].

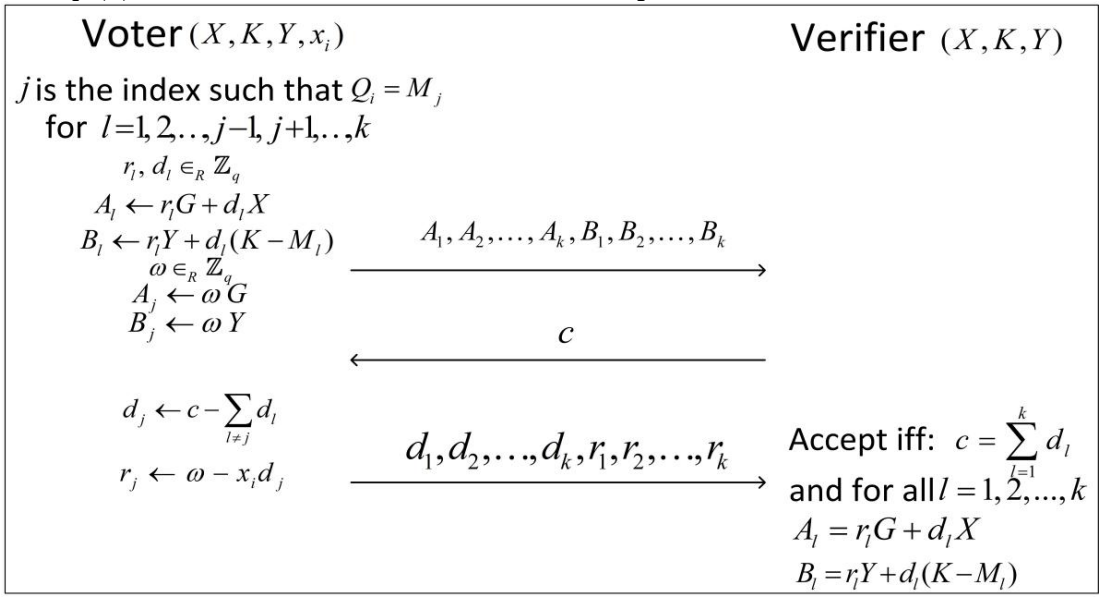

Figure 2. One-out-of-k proof of knowledge.

3.2 Voting Protocol Based on a Single Generator

Now we turn to another construction of multiple-candidate boardroom voting protocol using only a single generator. We still assumes there are k candidates and n eligible voters denoted $P_1, P_2, \ldots, P_n$. E Denotes an elliptic curve over a finite field $\mathbb{F}_q$ where the Decisional Diffie-Hellman problem is intractable, and G is a generator of E. H still denotes a publicly agreed collision-resistant hash function. Each voter P_i selects a value $x_i \in_R \mathbb{Z}_q$ at random as their private key. The voters execute the following three-round protocol:

Protocol 2.

Round 1. Each voter P_i broadcasts the voting key $x_i G$ and a non-interactive zero-knowledge proof $NIZKP(x_i)$ to prove the knowledge of the scalar x_i on the bulletin board. The proof $NIZKP(x_i)$ is generated by Schnarr's signature and Fait-Shamir heuristic as in Protocol 1. At the end of this round, every voter checks the validity of all zero-knowledge proofs and computes a set of reconstructed keys $Y_1, Y_2, ..., Y_n$ by Eq.(2). Suppose $Y_i = y_i G$ for some $y_i \in \mathbb{Z}_q$, then it holds that $\sum_i x_i y_i = 0$.

Round 2. Let m be the smallest integer in $\mathbb{Z}_q$ such that $2^m > n$. each voter P_i calculates the encrypted vote $x_i y_i G + v_i G$, where $v_i \in \{2^0, 2^m, 2^{2m}, ..., 2^{(k-1)m}\}$ and $v_i = 2^{(j-1)m}$ if P_i chooses the j^{th}

candidate. Then he commits his ballot by publishing the hash of the encrypted vote $H(x_i y_i G + v_i G)$.

Round 3. Each voter P_i publishes $x_i y_i G + v_i G$ and a non-interactive zero-knowledge proof $NIZKP(v_i)$ to prove that v_i is one of $\{2^0, 2^m, 2^{2m}, ..., 2^{(k-1)m}\}$ without leaking which one it is. This one-out-of-*k* zero-knowledge proof is generated in the same way as $NIZKP(Q_i)$ in Protocol 1, except that Q_i is replaced by $v_i G$, and $G_1, G_2, ..., G_k$ is replaced by $\{2^0, 2^m, 2^{2m}, ..., 2^{(k-1)m}\}$. all zero-knowledge proofs and commitments need to be verified before tallying.

Once all votes have been cast, anyone can compute $\sum_i (x_i y_i G + v_i G)$. Since $\sum_i x_i y_i = 0$, it is equal to $(\sum_i v_i)G$. its scalar to G is the sum of votes. The super-increasing property of the encoding guarantees that the sum can be unambiguously resolved into the count of votes for each candidate. Suppose

$$\sum_i v_i = c_1 2^0 + c_2 2^m + ... + c_k 2^{(k-1)m}, \tag{7}$$

Then $c_1, c_2, \ldots, c_k$ are the counts of votes for the *k* candidates respectively. As in the Protocol 1, to compute the tally, one needs to find the value $\sum_i v_i$ by exhaustive search. This can be sped up by pre-computing the possible combinations.

4. Boardroom Voting Protocol over Ethereum

In this section, we present how to implement a decentralized self-tallying voting by combining the Protocol 1 and Protocol 2 respectively with the Ethereum. The Ethereum block chain is used as the public bulletin board of the voting scheme, and its underlying peer-to-peer network serves as an authenticated channel for all voters. No trusted authority is required. The correct implementation of the voting protocol is enforced by the consensus mechanism of Ethereum. In addition to the voters, we still need an administrator to update the eligible voter list, set deadlines and choose parameters, but we stress that the administrator need not be a trustworthy authority. The way to implement Protocol 1 and Protocol 2 using Ethereum is quite similar. Thus we focus on the description of Protocol 1's implementation, and point out the execution of Protocol 2 when there are differences.

As mentioned earlier, self-tallying voting protocols have the drawback that the last voter is able to compute the tally ahead of others, which may lead to adaptive problems. Thus we require all voters to publish the commitment of their encrypted votes so that the last voter cannot change his ballot even though he is able to compute the tally. However, there remains abortive problems. The final voter can give up casting his vote if he is not satisfied with the tally. To address this issue, we require every voter to deposit a certain amount of ether when registering for voting. The money will be refunded as long as the voter completes the voting protocol.

The voting protocol is implemented in five stages and demands voter interaction only in three rounds. The five stages include: INITIALIZATION, SIGNUP, COMMIT, VOTE and TALLY, as shown in Fig 3. We describe each stage in details in the following:

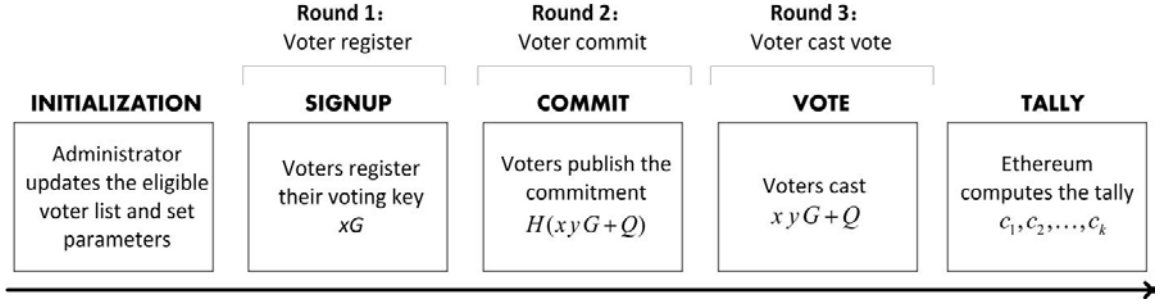

Figure 3. The five stages of the voting protocol in multiple-generator case

Intialization. The administrator sets the list of voting candidates and the amount of registration deposit d. then he authenticates every voter by their externally owned account and updates the voter list of the voting contract with these accounts. The administrator also sets a list of deadlines to guarantee that the voting is conducted in an orderly manner:

- $t_{EndRegistration}$: each voter P_i must register the voting key $x_i G$ before this time.

- $t_{EndCommit}$: each voter P_i must commit their vote before this time.

- $t_{EndVoting}$: each voter P_i must cast their ballot $x_i y_i G + Q_i$ (or $x_i y_i G + v_i G$ when Protocol 2 is implemented) before this time.

At last, the administrator informs Ethereum to move to the SIGNUP stage.

SIGNUP. Eligible voters can register in this stage. Each voter P_i computes the voting key $x_i G$ and the non-interactive proof $NIZKP(x_i)$ and sends them to Ethereum with d ether as a deposit. Ethereum only accepts registrations before $t_{EndRegistration}$. After all voters have registered, the administrator notifies Ethereum to compute all reconstructed keys $y_1 G, y_2 G, ..., y_n G$ and move to the COMMIT stage.

COMMIT. Each voter P_i sends their commitment $H(x_i y_i G + Q_i)$ (or $H(x_i y_i G + v_i G)$ when Protocol 2 is implemented) to the voting contract. The contract automatically enters the VOTE stage when all commitments are accepted.

VOTE. Each voter P_i sends their encrypted vote $x_i y_i G + Q_i$ (or $x_i y_i G + v_i G$ when Protocol 2 is implemented) and the one-out-of-k proof $NIZKP(Q_i)$ (or $NIZKP(v_i)$) to the voting contract. Ethereum refund he deposit once it receives the vote. After all votes are cast, the administrator notifies Ethereum to enter the TALLY stage.

TALLY. Ethereum verifies all one-out-of-k proofs and then computes the sum $\sum_i (x_i y_i G + Q_i) = \sum_i Q_i$, then searches for $c_1, c_2, ..., c_k \in \mathbb{Z}_q$ such that $\sum_i Q_i = c_1 G_1 + c_2 G_2 + \ldots + c_k G_k$. This is done by exhaustively calculating the sum of all C_{n+k-1}^{k-1} possible combinations. We present the pseudo-code for exhaustive search of $c_1, c_2, ..., c_k$ in Algorithm 1 below. Specifically, since k is a variable parameter selected by the administrator, we perform the exhaustive search recursively. Algorithm 1 defines a recursive function F, to compute the tally, one only needs to define a vector $(c_1, c_2, ..., c_k)$ of length n and initialize it to zero vector, then call function F with the arguments $m = k, \overline{n} = n, \overline{G} = \sum_i Q_i$.

If it is Protocol 2 that is implemented, Ethereum computes the sum $\sum_i (x_i y_i G + v_i G) = (\sum_i v_i) G$, then search for $c_1, c_2, ..., c_k$ such that $(c_1 2^0 + c_2 2^m + ... + c_k 2^{(k-1)m}) G = (\sum_i v_i) G$. This can be done by calculating all C_{n+k-1}^{k-1} possible combinations. Similar to the case of implementing Protocol 1, we can complete the exhaustive search by recursion. The pseudo-code is almost the same as Algorithm 1 and thus we omit it.

Algorithm 1 Compute Tally for Multiple-Generator Case. $F(m, \bar{n}, \bar{G})$.

Input: $m \in [0, k], \bar{n} \in [0.n], \bar{G} \in E$

Output: $c_1, c_2, \cdots, c_k$ such that $c_1 + c_2 + \cdots + c_k = \bar{n}$ and $c_1 G_1 + c_2 G_2 + \cdots c_k G_k = \bar{G}$

```
1: if m = 1 then
2:     if n̄G_1 = G_0 then
3:         return c_1, c_2, ··· , c_k;
4:     end if
5: else
6:     for a = 0, 1, 2, ··· , n̄ do
7:         c_m = a;
8:         F(m − 1, n̄ − a, Ḡ − aG_m);
9:     end for
10: end if
```

In the following we further elaborate subtle issues:

Deposit refund. After tallying, Ethereum will refund the deposits to all voters. There are also two scenarios where the voters can obtain their refund. First, if a registered voter has committed while some other voters not by $t_{EndCommit}$, he can claim his refund. Second, if a registered voter has voted while some other voters not by $t_{EndVoting}$, he can also claim his refund.

Generator construction. The *k* generators $G_1, G_2, ..., G_k$ used in Protocol 1 can be constructed in a simple way: choose a single G and compute $G_j \leftarrow H(encode(G) \| j)$ for $j = 1, 2, ..., k$, where H is some hash function. It is a usual method for producing multiple generators.

Replay attack. It is notable that an eligible voter P_i can register the same voting key $x_j G$ as some other voter P_j by replaying $x_j G$ and $NIZKP(x_j)$. This allows P_i latter copy the vote of P_j. Thus we in the above implementation, we require that the hash function's arguments must include the variable "msg. sender" when generating the non-interactive proof $NIZKP(x_j)$. This makes the protocol resistant to the replay attacks.

Scalar multiplication computation. During the execution of the Protocol 1 and Protocol 2, we need to perform scalar multiplication of points in the elliptic curve E frequently. The straightforward way to compute a scalar multiplication aG for $a \in \mathbb{Z}_q$ and $G \in E$ is to compute $G + G + ... + G$ with $a - 1$ additions. As a increases, the amount of calculation will increase greatly. One can use Square and multiply method to reduce the amount of calculation greatly. For example, when computing aG, one first resolves a into

$$a = b_k 2^k + b_{k-1} 2^{k-1} + ... + b_1 2 + b_0, \tag{8}$$

Then computes

$$aG = (((((b_k G) \times 2 + b_{k-1} G) \times 2 + b_{k-2} G) \times 2...) \times 2 + b_1 G) \times 2 + b_0 G. \tag{9}$$

This requires only $\sum_j b_j + k = O(\log a)$ additions. The pseudo-code of the *Square and Multiply* method is presented in Algorithm 2 below.

Algorithm 2 Scalar-Multiplication Method

Input: $G, (b_1, b_2, \cdots, b_k), k$
Output: $(\sum_{j=0}^{k} b_j 2^j)G$
1: **initialize**: Set $d = \mathbf{0} \in E$
2: **for** $j = k$ downto 0 **do**
3: $d = d + d$;
4: **if** $b_j = 1$ **then**
5: $d = d + G$;
6: **end if**
7: **end for**
8: **return** d;

5. Analysis

In this section, we analyze the protocol from the perspectives of security and efficiency, respectively. For the security, we first represent eight security conditions that a self-tallying voting scheme should meet, then prove that the boardroom voting protocol over Ethereum proposed in the previous section satisfies these conditions. For the efficiency, we analyze the computational complexity, communication complexity and payment of the protocol over Ethereum for both the multiple-generator case and the single-generator case, and compare these two kinds of implementation.

5.1 Security Analysis

To analyze the security of the protocol, we first need to consider the attack model. There are two kinds of adversary. Passive adversaries only eavesdrop on the communication, while the active adversaries participate in the vote and may collude with other voters. Any decentralized voting scheme cannot resist a full collusion where all voters are dishonest except one, since adversaries can compute the vote of the only honest voter by simply subtracting their votes from the tally. Therefore, we only consider the partial collusion.

As mentioned in [13, 14], self-tallying anonymous voting protocols under the partial-collusion attack model should satisfy three requirement: maximum ballot secrecy, self-tallying and dispute-freeness. Simultaneously, a general voting scheme also needs to fulfill seven basic properties, which is proposed by A. Fujioka *et al.* [19] and have been widely used. Excluding two repeated requirements, we obtain eight secure properties that a self-tallying voting protocol needs to satisfy:

-Completeness: each valid ballot is correctly counted.

-Robustness (soundness): dishonest voters cannot disrupt the voting.

-Eligibility: only eligible voters are able to vote.

-Unreusability: no voter is able to vote twice.

-Fairness: Each voter casts his vote independently and cannot vote depending on other honest voters' choices.

-Maximum ballot secrecy: every cast ballot must be a cipher text indistinguishable from random, and therefore reveal nothing about the choice of the voter.

-Self-tallying: After all votes have been cast, anyone is able to compute the result by himself.

-Dispute-freeness (verifiability): anyone can verify whether all voters behave according to the protocol.

In the following we demonstrate that the protocol presented in the fourth section fulfills the above eight properties. It is obvious to see that the completeness, eligibility, unreusability and self-tallying are satisfied. Thus we focus on the remaining four properties.

Robustness. The protocol's robustness is guaranteed by the liveness of the underlying block chain and the financial incentive introduced by the deposit-refund paradigm. If a voter correctly implements the protocol, he will interact with the smart contract on Ethereum by initiating transactions. The liveness property of the Ethereum block chain ensures that the transaction will be

included in a block of the block chain soon, so that the ballot of the voter must be cast. Therefore, the attacker cannot disrupt the vote of honest voters. As for the dishonest eligible voters, the financial incentive provided by the deposit-refund paradigm will enforce them to cast their ballots in time.

Maximum ballot secrecy. We only prove the maximum ballot secrecy of the protocol for the multiple-generator case. The proof for the single-generator case is similar. In the protocol, the adversary can see the voting key $x_i G$, the encrypted ballot $x_i y_i G + Q_i$, its commitment $H(x_i y_i G + Q_i)$ and two non-interactive zero-knowledge proofs. Due to the security of the hash function, the commitment does not reveal any information of the ballot. Therefore, we only need to proof that $x_i y_i G + Q_i$ is indistinguishable from random to the adversaries in a partial collusion attack even given $x_i G$ and the proofs $NIZKP(x_i), NIZKP(Q_i)$. Specifically, we have following theorem:

Theorem 1. Under the Elliptic Curve Diffie-Hellman assumption, adversaries in a partial collusion attack against a voter P_i cannot distinguish the encrypted ballot $x_i y_i G + Q_i, Q_i \in \{G_1, G_2, ..., G_k\}$ from a random point in the elliptic curve group E.

Proof. In addition to the encrypted ballot, adversaries can also obtain $x_i G, NIZKP(x_i)$ and $NIZKP(Q_i)$. Due to the security of the Schnarr's signature, Fait-Shamir heuristic and CDS technique, both non-interactive proofs will not reveal any information other than the assertions proved. Thus we only need to consider the voting key $x_i G$. The secret key x_i is chosen randomly by the voter P_i. According to the Elliptic Curve Diffie-Hellman assumption, as long as y_i is randomly and secretly chosen from $\mathbb{F}_q$, the encrypted ballot $x_i y_i G$ will be indistinguishable from a random element in E. Hence the only thing left is to prove that y_i is a secret random value. To prove this, consider the worst situation where there is only one voter $P_l (l \neq i)$ other than P_i participates in the collusion. Then its secret key x_l in uniformly distributed in $\mathbb{F}_q$. Since y_i is computed by doing addition or subtraction between all $x_j (j \neq l, i)$ and x_l, y_i is also a uniformly random element in $\mathbb{F}_q$.

Theorem 1 states that each encrypted ballot is indistinguishable from a random element in the elliptic curve group to any adversary in a partial collusion attack, which means the protocol fulfills the maximum ballot secrecy.

Fairness. The fairness requires that each voter casts his vote independently and cannot vote depending on other honest voters' choices. For all eligible voter who are not the last to vote, the ballots of other voters are all random elements in the elliptic curve due to the maximum ballot secrecy property. Thus they are not able to vote according to the choices of other voters. Moreover, the commitment stage enforce that the last voter cannot change his vote even though he can compute the tally before others. This ensures that even the last voter cannot decide his vote according to other voters' choices.

Dispute-freeness. The underlying peer-to-peer network of Ethereum allows anyone to authenticate the identity of the voters, and by verifying the non-interactive zero-knowledge proofs, everybody can check whether the ballots cast by the voters are valid. These verification can guarantee that all voters execute the protocol honestly. Due to the self-tallying property, these verification ensures the correctness of the final tally.

5.2 Efficiency and Comparison

We analyze the efficiency of the protocol from the perspectives of computational complexity, communication complexity and payment. The computational complexity includes two parts: local calculations performed by the administrator or voters; the computations initiated by the administrator or votes but executed on the Ethereum Virtual Machine (EVM) by the miners. The computations on the EVM is expensive and paid by gas, thus they are the main considerations of the

computational complexity. Communication complexity refers to the total length of messages that the administrator or voters need to send to the smart contracts. Ethereum nodes uses the "data" field in the transactions to transmit messages to smart contracts. The larger the data transmitted, the more gas spent. The gas payment depends on both the computational complexity and the communication complexity. We will define the gas payment of several basic operations, then estimate the total payments of the administrator and the voters for the multiple-generator case and single-generator case respectively.

We first consider the efficiency of the administrator. To this end, we analyze in detail the operations that the administrator need to execute at each stage. At the stage INTIALIZATION, the administrator needs to send the list of n eligible voters, sets the timers and notifies the smart contract to enter the last stage. All these can be done through one transaction (e.g. define a function to contain all these operation in the smart contract). Each address over Ethereum have 160 bits. Thus the eligible-voter list has a total of $160n$ bits. The length of the timers is small compared to the list, so we omit it in the analysis for simplicity. Therefore, the communication complexity in this stage is $160n$ bits. Suppose each bit of data transmission requires f_{data} gas and sending a basic transaction costs β gas (which is 21,000 at the time of writing), then the total payment of the administrator in this stage is about $(160nf_{data}+\beta)\alpha$ ETH, where α is the gas price measured in ether.

In the SIGNUP stage, the administrator need to inform the smart contract to enter the next stage. Meanwhile, the transaction initiated by the administrator also call the function of the smart contract to compute the reconstructed keys and verify the non-interactive proofs $NIZKP(x_i)'s$. These computations will finally be executed on the EVM by the miners. Generating each reconstructed key requires $n-2$ additions of group elements. Thus computing all reconstructed keys needs $n(n-2)$ group additions. To estimate the amount of calculations required for verifying the zero-knowledge proofs, we first analyze how many group additions is needed to calculate a random scalar multiplication on average. By the *Square and Multiply* method, compute a scalar multiplication xG requires $O(\log x)$ group additions. Thus, for a randomly chosen $x \in \mathbb{F}_q$, the average addition needed to compute xG is

$$\frac{1}{q}\sum_{x=1}^{q}\log x=\frac{1}{q}\log(q!)\xrightarrow{String\ formula}O(\frac{1}{q}\log(\sqrt{2\pi n}(\frac{n}{e})^n))=O(\log q), \qquad (10)$$

That is, calculate a random scalar multiplication requires $O(\log q)$ additions on average. Verifying each proof $NIZKP(x_i)$ needs one hash computation, two random scalar multiplications and one group addition. Thus $O(2\log q+1)$ group additions and one hash computation is required in total. In summary, the computational complexity of this stage contains one hash computation and $O(2\log q+n(n-2)+1)$ group additions on EVM. Suppose one group addition requires f_{add} gas, one hash computation requires f_{hash} gas, then the total payment is about $(2\log q+n(n-2)+1)f_{add}\alpha+f_{hash}\alpha+\beta\alpha$ ETH.

The administrator does not participate in the COMMIT and the VOTE stages. But in the end of the VOTE stage, the administrator need to notify Ethereum to enter the TALLY stage. The transaction initiated by the administrator will then call the function of the smart contract to verify all one-out-of-*k* zero-knowledge proofs and compute the tally. Corresponding computations will then be executed by the miners. The computational complexity of this stage is different in the multiple-generator case and the single-generator case. We first consider the multiple-generator case. As shown in Fig.2, verifying each $NIZKP(Q_i)$ needs to compute $A_1,...,A_k,B_1,...,B_k$. Each A_l requires two random scalar multiplications and one group addition, while each B_l requires two random scalar multiplications and two group additions. Thus computing $A_l's$ and $B_l's$ requires a total

$O(4k\log q+3k)$ group additions. In addition, verifying the proof also needs one hash computation and *k*-1 additions in $\mathbb{F}_q$, so the total computation needed for verifying the *n* one-out-of-*k* zero-knowledge proofs includes $O(4kn\log q+3kn)$ group additions, $(k-1)n$ additions in $\mathbb{F}_q$ and n hash computations. To compute the tally, the EVM should first compute the sum $\sum_i (x_i y_i G+Q_i)$, which needs *n*-1 group additions, then searches for $c_1,c_2,...,c_k \in \mathbb{Z}_q$ by exhaustively calculating the sum of all C_{n+k-1}^{k-1} possible combinations. Computing $c_1G_1+c_2G_2+\ldots+c_kG_k$ for random c_i's requires $O(k\log q+k-1)$ group additions, so the exhaustive search needs $O(C_{n+k-1}^{k-1}(k\log q+k-1))$ group additions. This stage's computational complexity contains $O(C_{n+k-1}^{k-1}(k\log q+k-1)+4kn\log q+3kn)$ group additions, $(k-1)n$ additions in $\mathbb{F}_q$ and n hash computations. Suppose one addition in $\mathbb{F}_q$ requires f_{add-q} gas, the total payment is $(C_{n+k-1}^{k-1}(k\log q+k-1)+4kn\log q+3kn)f_{add}\alpha+nf_{hash}\alpha+(k-1)nf_{add-q}\alpha+\beta\alpha$ ETH.

For the single-generator case, verifying each $NIZKP(v_i)$ requires to compute $2^m G, 2^{2m}G..., 2^{(k-1)m}G$ in addition. This needs $O(0.5k(k-1)\log n)$ group additions. By the Square and Multiply method, the exhaustive search for $c_1,c_2,...,c_k$ only needs $O(C_{n+k-1}^{k-1}k\log n)$ group additions. Therefore, for single-generator case this stage needs a total of $O(C_{n+k-1}^{k-1}k\log q+4kn\log q+3kn+0.5k(k-1)\log n)$ group additions, $(k-1)n$ additions in $\mathbb{F}_q$ and n hash computations. The total payment is $(C_{n+k-1}^{k-1}k\log q+4kn\log q+3kn+0.5k(k-1)\log n)f_{add}\alpha$ $+nf_{hash}\alpha+(k-1)nf_{add-q}\alpha+\beta\alpha$ ETH.

Table 1. Gas cost for various basic operations.

Operation	Gas cost
Sending a transaction	β
Data transmission per bit	f_{data}
Hash computation	f_{hash}
Group addition	f_{add}
Addition in $\mathbb{F}_q$	f_{add-q}

In summary, we list the gas cost of each basic operation in Table 1, and present the total computational complexity, communication complexity and payment of the administrator for the whole implementation in Table 2, where $\Gamma(k,q,n)$ and $\Lambda(k,q,n)$ are defined as:

$$\Gamma(k,q,n)=C_{n+k-1}^{k-1}k\log q+kn(4\log q+3)+n^2-n+2\log q+1, \tag{11}$$

$$\Lambda(k,q,n)=\Gamma(k,q,n)f_{add}\alpha+(n+1)f_{hash}\alpha+(k-1)nf_{add-q}\alpha+3\beta\alpha. \tag{12}$$

Table 2. Administrator's efficiency of the entire implementation.

Efficiency / Underlying protocol		Multiple-generator protocol	Single-generator protocol
Computational Complexity	Group addition (on EVM)	$\Gamma(k,q,n)+C_{n+k-1}^{k-1}(k-1)$	$\Gamma(k,q,n)+\frac{1}{2}k(k-1)\log n$
	Hash computation (on EVM)	$n+1$	$n+1$
	Addition in $\mathbb{F}_q$ (on EVM)	$(k-1)n$	$(k-1)n$
Communication complexity (bit)		$160n$	$160n$
Payment (ETH)		$\Lambda(k,q,n)+C_{n+k-1}^{k-1}(k-1)f_{add}\alpha$	$\Lambda(k,q,n)+\frac{1}{2}k(k-1)\log nf_{add}\alpha$

Now we turn to the voter's efficiency. The voters do not participate in the INTIALIZATION and TALLY stages. In the SIGNUP stage, each vote needs to compute the voting key x_iG and the non-interactive proof $NIZKP(x_i)$ locally. Computing the voting key requires one random scalar multiplication, while the non-interactive proof requires one hash computation, one addition in $\mathbb{F}_q$, one multiplication in $\mathbb{F}_q$ and one random scalar multiplication. Thus, in this stage each voter's computational complexity contains local computations of $O(2\log q)$ group additions, one hash computation, one addition in $\mathbb{F}_q$ and one multiplication in $\mathbb{F}_q$. each voter needs to send the voting key and the proof to the smart contract, they contain two elements in the elliptic curve group and an element in $\mathbb{F}_q$ in total. Ethereum uses the elliptic curve digital signature algorithm Keccak256. In order to be consistent, we assume the elements of the elliptic curve group E also have 256 bits. Thus the communication complexity of this stage is $512+\log q$ bits. Since local computations do not cost gas, the payment is about $(512+\log q)f_{data}\alpha \quad +\beta\alpha$ ETH.

In the COMMIT stage, we first consider the multiple-generator case. Each voter computes the encrypted vote $x_iy_iG+Q_i$ and its commitment $H(x_iy_iG+Q_i)$, this requires local computations of one random scalar multiplication, one group addition and one hash computation. Thus the computational complexity of each voter in this stage contains one local hash computation and $O(\log q+1)$ local group additions. Each voter needs send a hash to the smart contract. The hash used in Ethereum has a 160-bit output, thus the communication complexity is 160 bits. The total payment is $160f_{data}\alpha+\beta\alpha$. For the single-generator case, each voter also needs to compute v_iG for some $v_i\in\{G,2^mG,...2^{(k-1)m}G\}$ besides the above operations. Since they need to compute v_iG for all $v_i=1,2^m,...2^{(k-1)m}$ in the next stage, we ignore this part of calculation at this stage. Therefore, the efficiency of each voter in the single-generator case is the same as in the multiple-generator case.

In the VOTE stage, we first consider the multiple-generator case. Each voter needs to compute the non-interactive proof $NIZKP(Q_i)$ and send it with the encrypted vote $x_iy_iG+Q_i$ to the smart contract. The computational complexity contains $O(4k\log q+2\log q+3k)$ local group additions and one local hash computation. The proof $NIZKP(Q_i)$ contains $2k$ elements in the elliptic curve group and $2k$ elements in $\mathbb{F}_q$, thus the communication complexity is $2k\log q+256(2k+1)$ bits. Since local computations do not cost gas, the total payment is about $(2k\log q+512k+256)f_{data}\alpha+\beta\alpha$ ETH. For the single-generator case, each voter has almost the same efficiency except that they need additional $O(0.5k(k-1)\log n)$ group additions.

Table 3. Each voter's efficiency of the entire implementation.

Efficiency / Underlying protocol		Multiple-generator protocol	Single-generator protocol
Computational Complexity	Group addition (on local computer)	$\Delta(k,q)$	$\Delta(k,q)+\frac{1}{2}k(k-1)\log n$
	Hash computation (on local computer)	3	3
	Addition in $\mathbb{F}_q$ (on local computer)	1	1
	Multiplication in $\mathbb{F}_q$ (on local computer)	1	1
Communication complexity		$(2k+1)\log q+512k+928$	$(2k+1)\log q+512k+928$
Payment (ETH)		$\Theta(k,q)\,f_{data}\alpha+3\beta\alpha$	$\Theta(k,q)\,f_{data}\alpha+3\beta\alpha$

In summary, we present the total computational complexity, communication complexity and payment of each voter for the whole implementation in Table 3, where they $\Delta(k,q)$, $\Theta(k,q)$ are defined as:

$$\Delta(k,q)=4k\log q+5\log q+3k+1, \tag{13}$$

$$\Theta(k,q)=(2k+1)\log q+512k+928. \tag{14}$$

Through the above analysis, we can see that the main cost of the administrator is for the calculations on the EVM, while the main cost of the voters is for transmitting data to the smart contract. By comparison, the multiple-generator case and single-generator case have the same communication complexity. With respect to the computational complexity, the single-generator case requires more local computations, while the multiple-generator case requires more computations on the EVM. Since the computations on the EVM is much more costly and consumes gas, the single-generator case has better computational complexity and costs less ethers. In addition, the multiple-generator case also require the smart contract to store additional k generators. The storage of the EVM is very expensive, this makes the deploy of the voting smart contract more expensive in the multiple-generator case than in the single-generator case.

6. Conclusion

In this paper, we present a decentralized anonymous voting protocol that is self-tallying and support multiple-candidate based on Ethereum smart contracts. We provide two specific ways to implement the protocol over Ethereum. One uses multiple generators of an elliptic curve group while the other one uses only a single generator. The Ethereum block chain serves as the public bulletin board and the underlying peer-to-peer network is used as an authenticated channel. The consensus mechanism that guarantees the security of Ethereum enforces the correct implementation of the protocol and guarantees its security. Both implementation methods are proved to fulfill completeness, robustness, eligibility, unreusability, fairness, maximum ballot secrecy and dispute-freeness under the partial-collusion attack model. By the efficiency analysis of both implementations, we demonstrate that the main cost of the administrator is for the calculations on the EVM, while the main cost of the voters is for transmitting data to the smart contract. The multiple-generator case and single-generator case have the same communication complexity, but the single-generator case has better computational complexity and costs less ethers.

Acknowledgments

This research was funded by National Natural Science Foundation of China (Grant No. 61672517) and National Cryptography Development Fund (Grant No. MMJJ20170108).

References

[1] S. Nakamoto, “Bitcoin: a peer-to-peer electronic cash system” (2008).

[2] J. Garay, a Kiayias, N Leonardos, “The bitcoin backbone protocol: Analysis and applications”. In: *Advances in Cryptology - EUROCRYPT 2015* (Springer, Sofia, Bulgaria, 2015), pp. 281 - 310.

[3] G. Wood, “Ethereum: a secure decentralized transaction ledger” (2014).

[4] J. Bonneau, “EthIKS: Using Ethereum to audit a CONIKS key transparency log”. In: *International Conference on Financial Cryptography and Data Security* (Springer, Rome, Italy, 2016), pp. 95 - 105.

[5] M. Al-Bassam, “SCPKI: a smart contract-based PKI and identity system”. In: *ACM Workshop on Blockchain, Cryptocurrencies and Contracts* (ACM, Abu Dhabi, UAE, 2017), pp. 35 - 40.

[6] C. Dong, Y. Wang, A. Aldweesh, et al., “Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing”. In: *2017 ACM SIGSAC Conference on Computer and Communications Security* (ACM, Toronto, Canada, 2017), pp. 211 - 227.

[7] B. Xia, D. Ji, G. Yao, “Enhanced tls handshake authentication with block chain and smart contract”. In: *International Workshop on Security, IWSEC 2017* (Springer, Hiroshima, Japan, 2017), pp. 56-66.

[8] P. McCorry, S. F. Shahandashti, F. A. Hao, “smart contract for boardroom voting with maximum voter privacy”. In: *International Conference on Financial Cryptography and Data Security* (Springer, Sliema, Malta, 2017), pp. 357 - 375.

[9] F. Zhang, E. Cecchetti, K. Croman, et al., “Town crier: An authenticated data feed for smart contracts” In: *2016 ACM SIGSAC conference on computer and communications security* (ACM, Vienna, Austria, 2016), pp. 270 - 282.

[10] S. Amani, M. Bégel, M. Bortin, et al., “Towards verifying ethereum smart contract bytecode in Isabelle/HOL”. In: *The 7th ACM SIGPLAN International Conference on Certified Programs and Proofs* (ACM, Los Angeles, California, United States, 2018), pp. 66 - 77.

[11] R. CRAMER, R. GENNARO, B. SCHOENMAKERS, “A secure and optimally efficient multi-authority election scheme”. In: *EUROCRYPT 1997,* (Springer, Konstanz, Germany, 1997), pp. 103-118.

[12] R. CRAMER, M. FRANKLIN, B. SCHOENMAKERS, M. YUNG, “Multiauthority secret-ballot elections with linear work”. In: *EUROCRYPT 1996* (Springer, Saragossa, Spain, 1996), pp. 72 – 83.

[13] A. KIAYIAS, M. YUNG, “Self-tallying elections and perfect ballot secrecy”. In: *Public Key Cryptography, PKC 2002* (Springer, Paris, France, 2002), pp. 141 - 158

[14] J. GROTH, “Efficient maximal privacy in boardroom voting and anonymous broadcast”. In: *International Conference on Financial Cryptography’ 04* (Springer, Key West, FL, USA, 2004), pp. 90 - 104.

[15] F. Hao, P.Y.A. Ryan, P. Zielisńki, “Anonymous voting by two-round public discussion”. *IET Information Security*, 4(2), 62-67 (2010).

[16] C. P. SCHNORR, “Efficient signature generation by smart cards”. *Journal of Cryptology,* 4(3), 161-174 (1991)

[17] A. Fiat, A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems”. In: *CRYPTO’ 86* (Springer, Santa Barbara, CA, USA, 1986), pp. 186 - 194.

[18] J. GROTH, R. CRAMER, I. Damgård, B. SCHOENMAKERS, “Proofs of partial knowledge and simplified design of witness hiding protocols”. In: *CRYPTO’ 94* (Springer, Santa Barbara, CA,

USA), pp. 174 - 187.

[19] A. Fujioka, T. Okamoto, K. Ohta, “A practical secret voting scheme for large scale elections”. In: *International Workshop on the Theory and Application of Cryptographic Techniques* (Springer, 1992), pp. 244 – 251.