A Privacy-Preserving Voting Protocol on Blockchain

Wenbin Zhang[‡]
Ant Financial, Hangzhou, China
zh.wb@outlook.com

Sheng Huang
UMark Co. Ltd, Shanghai, China
huangsheng@umarkcloud.com

Yuan Yuan[*], Yanyan Hu, Shaohua Huang, Shengjiao Cao, Anuj Chopra
IBM Center for Blockchain Innovation
IBM Research, Singapore
idayuan, yanyanhu, huang, caosj, achopra@sg.ibm.com

***Abstract*—As blockchain technologies mature and ecosystems over blockchain evolve, peers on blockchain networks often face situations in which they need to conduct voting for decision-making; as happened in the case of the DAO hard fork event on Ethereum. However, a natively built-in voting mechanism is not available on any of the existing blockchain platforms. Thus, the decision making either is delegated to a few network members who make such decisions offline or is dependent on third party online voting services. In both cases, peers directly or indirectly rely on trusted parties or centralized systems. This is against the basic decentralization principle of blockchain and exposes the election to frauds. To facilitate decision-making in a decentralized and secure manner, we propose a native blockchain voting protocol for peers to vote over their existing blockchain network without the need of any trusted or third party. Our protocol preserves end-to-end privacy and possesses desirable properties such as detectability and correctability against cheating. A reference implementation of our protocol on Hyperledger Fabric that demonstrates the validity and practical applicability of our protocol is also provided.**

Keywords—Blockchain; voting; distribution voting; distributed tally

I. Introduction

The interest in blockchain as a technology increased tremendously after the original Bitcoin paper [1] was published in the year 2008 by Satoshi Nakamoto. The attention to the potential of the technology reached a peak when Ethereum [2] was announced to be a Turing-complete platform thereby enabling the capability to perform any complex logic operation on blockchain network. Subsequently, many interesting and revolutionary applications were developed on the Ethereum platform.

Among others, The DAO, a distributed autonomous organization was formed. It inaugurated a new era of manager/leader-less corporations and was a form of investor directed venture capital fund. The DAO became the largest crowdfunded organization in history, having raised over $150m in 4 weeks, through a token sale in May 2016. However, right after the fund raising event, there was a cyber-attack on the DAO network resulting in $60m worth of ether stolen. This attack put Ethereum foundation and Ethereum network runners (miners) into a dilemma: whether to accept the attack or to create a fork to invalidate the attack transaction and hence mitigate the impact. To fork or not to fork, both decisions had pros and cons. The Ethereum foundation wanted the miners to make a decision which required a voting to take place among the miners. However, Ethereum platform didn't had a way to facilitate this voting event. Miners had to either rely on their mining pool's ad-hoc voting process managed by mining pool admin [5] or use a community initiated voting platform such as Carbonvote [6] to signal their intention. The latter could be done by sending "sudo ethers" to two third-party controlled addresses for tally. However, Carbonvote was just an "informal signaling tool", as quoted by Ethereum founder Vitalikk Buterin, and questions were raised on whether it was even an appropriate voting tool to get a fair vote from the miners who wished to participate.

As the technology matures, we expect to see more and more real-world applications using Blockchain. We foresee that there will be more voting events in the future and hence there is an exigent need for a native voting mechanism built directly on the blockchain network in the spirit of decentralization and disintermediation of the network itself. We also feel that the need for such a voting mechanism is not limited to public blockchain networks. It's there for consortium permissioned blockchain networks, such as Hyperledger Fabric [7], as well. Typically, in permissioned blockchains, there is no notion of mining or miner and the network runners are called peers. To avoid confusion, we use the term peer voting to represent both miners voting on public blockchain or peers voting on permissioned blockchain.

Voting is a universal phenomenon and is part of various societies in one way or the other. However, peer voting differs from voting activities like presidential elections. It is typically done online rather than traditional (physical) voting and thus introduces different kinds of challenges. As summarized in [8], an ideal online voting system would have many desirable characteristics. The most important ones are:

Eligibility: Only authorized voters can vote. In peer voting, we restrict voters to be miners on public blockchain network, peer participants on consortium blockchain. In both cases, voters would have already been authenticated by the underlying network. Sybil attack is not a concern here.

Integrity: Votes should not be modified, forged, or deleted without detection.

[‡] Most of this work was done while the author was with IBM Center for Blockchain Innovation, IBM Research, Singapore, until April 2018.

[*] Corresponding Author

2159-6190/18/$31.00 ©2018 IEEE
DOI 10.1109/CLOUD.2018.00057

IEEE computer society

Auditability: It should be possible to verify that all votes have been correctly accounted in the tally.

On top of these basic requirements, peer voting on blockchain should also satisfy the following requirements to be applicable in real world voting events, namely:

End-to-End Privacy Preserving: Voter's vote should not be revealed to any one at any step.

Detectability and Correctability. Invalid or malicious votes can be detected and excluded from tally, and any dishonest tally can be corrected.

No Trusted Authorities Needed. This is dictated by the properties of a blockchain network, namely decentralization and disintermediation.

To conduct peer voting, there is no well-designed mechanism available yet. Majority of existing online voting mechanisms do not satisfy all of the above-mentioned requirements. Many of these mechanisms fail due to dependency on trusted authorities for tally or cheating detection/correction. There are no easy extensions possible as well. We will share a detailed analysis on these existing options, later in the literature review part.

To fill in the gap, we propose a privacy preserving voting protocol that does not require any trusted party while providing the desired properties for peer voting mentioned above. The key ideas consist of *1) Distribution Voting*. Instead of one ballot per peer, we assign multiple ballots to a peer. The peer's voting intention is defined by the mode of his/her ballots set. This design ensures that the privacy of the vote is preserved. The voting intention can only be revealed if majority of voting ballots set is exposed. It cannot be revealed by exposing a single ballot. *2) Distributed Tally*. With the application of homomorphic encryption, our protocol can distribute the counting of ballots to each peer while ensuring no peer can gain knowledge of the final voting result. Due to this design, we eliminate the need for a trusted party for tally. *3) Cryptography-based verification*. The verification of votes and tally guarantees the detection of dishonest behaviors of voters throughout the protocol. Moreover, since the verification can be done publicly without breaching privacy, no trusted party is needed either.

The paper is organized as follows: Section II gives literature review of online voting techniques and gaps in their application to peer voting. Then Section III details our proposed protocol followed by an analysis of important properties of our protocol in Section IV. Section V presents the implementation details of the protocol on Hyperledger Fabric. Section VI concludes the paper with potential future work.

II. Literature Review

An online voting system has been a hot topic in literature for long. Many multi party voting algorithms have been proposed over the years to preserve the secrecy of voting information and thus the integrity of voting in an online format. As highlighted by [9], privacy of a vote may be preserved in two ways, by encrypting the vote or by sending the vote through an anonymous communication channel. The first approach of encrypting the vote was proposed in [10], [11], [12], [13]. But all these implementations were a bit impractical due to the amount of communication needed in order to validate the votes. The second approach was proposed in [9], [14], and [15]. This approach was more practical in terms of amount of communication needed but suffered from other challenges such as dependency on a central authority to tally the votes and possibility of ballot collision due to usage of random strings to distinguish each voter's ballot [16].

Reference [17] proposed a non-interactive verifiable secret sharing scheme using homomorphic encryption techniques, that could be used to verify the votes publicly while preserving the voter privacy. But this again required a central authority to be present that would be able to decrypt the votes to match the final tally. One common problem with the above schemes was that the user still had a receipt which could be made public hence affecting the vote privacy. Thus, various receipt-free voting schemes were also presented [18], [19]. [20] presented a simple multi-authority election scheme in which a voter posts a single ballot (an encrypted message) along with a proof that the ballot contains a valid vote.

Eventually, many online voting platforms, e.g. HELIOS [21], also emerged that made use of the web technologies to facilitate online voting. Such voting platforms, even though were not catered for high stake elections, had significant shortcomings in terms of security and user privacy as highlighted by [22], which demonstrates how the voting results can be compromised using existing web vulnerabilities on the client side. Such online voting systems rely on a centralized authority which becomes a single point of failure and in the event of breach may lead to a compromise in secrecy of votes, voter identity/privacy, service availability for voting and thus integrity of the results. [23] also demonstrated how an election officer with malicious intent can rig an election through an arbitrary result by issuing cryptographic proofs that the results were correctly tallied.

Since bulk of the online voting happens on a public domain, [24] proposed a masked ballot voting technique for receipt-free online elections by using a secret mask to disguise each user's vote. Reference [25] proposed a distributed, privacy-preserving and end-to-end verifiable e-voting system called D-DEMOS that does not use any cryptographic operations on the client side. This is done using a distributed vote-collection subsystem that collects the user votes and sends it to a bulletin board which is another distributed component. D-DEMOS adopted the Chaum-Pedersen zero-knowledge proofs [26] that allow the election authority to show that the vote inside a ballot is valid. However, this solution was also dependent upon a trustee subsystem that holds the encryption keys to access the data stored in the Bulletin Board component which can be breached. [27] proposed a distributed version of the HELIOS system.

Reference [28] proposed a practical multi-party computation algorithm for a secure distributed online voting system in which the election should be divided into small clusters of voters and each cluster participate in partial elections and the results are later joined together to get the overall election results. We are using a similar approach along with homomorphic encryption in our solution. With the rise in popularity of the distributed ledger or the blockchain technology, it was proposed that blockchain could be used to facilitate online elections given that a blockchain ledger is publicly verifiable. Reference [29] proposed a blockchain based voting platform to conduct national level elections. Their solution was also dependent upon a trusted third party to hide the user's vote from the election organization.

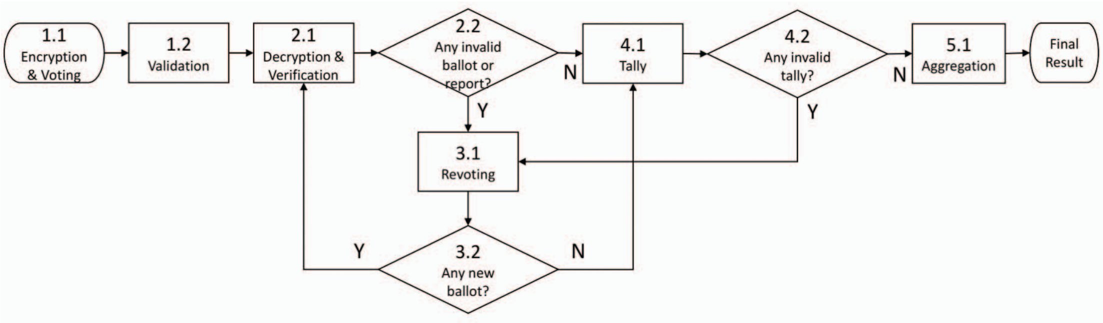

Figure 1. Process flow of the voting protocol

One big problem in this approach was that if the trusted third party can get the voter identification id, it can easily map all votes to the voters thus the confidentiality of the votes can be breached. Reference [30] also proposes the design of a distributed online voting system using blockchain technology as a substitute of the various central authorities. It also highlights that the problems of traditional online voting are still existing in the current blockchain based voting solutions available in the market.

So far, most of the blockchain based electronic platforms were focused on facilitating certain kind of general elections online. During events when the community members need to take some decisions and need to conduct a voting, they rely on existing online voting platforms. None of the current blockchain platforms provide a way to facilitate voting for the existing peers on the network. In this paper, we propose a receipt-free, perfectly verifiable and privacy preserving peer voting protocol that can help facilitate voting for peers existing on a blockchain network. No trusted third parties are required for voter identification or votes tallying. Our protocol leverages the cryptographic primitives using public key cryptography and homomorphic encryption [17], [18], [19] and relies on the consensus mechanism, which is inherent to the blockchain technology, to facilitate peer voting on blockchain while meeting all the requirements for a trusted and fair voting event. We also used an approach as outlined in [24], [25] to achieve distributed voting and distributed tally to preserve voter privacy while retaining public verifiability.

III. On-Chain Peer Voting

In this section, we will start by explaining the core idea of the protocol design while setting some technical assumptions. Later, we will continue by providing the technical details.

A. Idea of the Protocol Design

While detecting and correcting dishonest activities during peer voting, the primary challenge is to preserve privacy in the absence of a trusted party. To overcome this challenge, we propose the following two ideas which are key to our protocol design[1]:

I. *Distribution Voting*. Each vote is represented as a distribution, by being divided into many parts and distributed to many randomly selected peers. Each part can be verified independently and a small collection of these do not reveal the overall vote preference.

II. *Distributed Tally*. Tallying of votes is distributed among all peers, with each peer tallying a small portion. Each peer's tally can be verified and corrected, and all correct partial tallies will be aggregated to get the final voting result.

B. Assumptions

We prepare some notations and assumptions before the detailed protocol description. Assume there are m options for n voters to vote and each voter has K ballots B_k^j $(1 \leq k \leq K)$. Let N be an integer bigger than the total number of ballots nK for tallying ballots. We denote for any integer x a base-N numeral representation, $x = \sum \beta_i(x) N^i$, where $\beta_i(x)$ denotes the number in the ith digit. When $N = 10$, this is the commonly known decimal number notation.

1) Voters: Assume that there are n peers, P_j $(1 \leq j \leq n)$, in a blockchain network who can participate in voting[2]. The total number of voters will not increase during the voting process.

2) Choice: The choice C of an option in a ballot B is considered as valid if it is of the following representation,

$$C = N^{i-1} + r \times N^m, 1 \leq i \leq m,$$

where (i) N^{i-1} represents the i-th option, and (ii) r is a random number for the purpose of protecting the privacy of the choice after encryption.

3) Homomorphic Encryption: A homomorphic encryption scheme[3] is used to encrypt the choice in each ballot. Each voter

[1] Note that the security foundation of Blockchain relies on both cryptography and consensus. The even more fundamental idea behind the two key ideas is that some of the difficulty is offloaded to the consensus of Blockchain.

[2] For permissioned network, peers are those participants who hold peer nodes; while in permissionless network, like Ethereum, peers are the miners.

[3] A deterministic homomorphic encryption is needed here, but a probabilistic homomorphic, such as the Paillier encryption scheme, can be treated as deterministic by fixing its random number.

P_j generates his private key HESK$_j$ and public key HEPK$_j$. For each ballot B_k^j , the encryption of the choice C_k^j is,

$$EC_k^j = Encypt(HEPK_i, C_k^j)$$

where $i \neq j$. It has the following homomorphic property

$$Encypt\left(HEPK_i, C_k^j\right)\times Encypt\left(HEPK_i, C_{k\prime}^{j\prime}\right) = Encrypt(HEPK_i, C_k^j + C_{k\prime}^{j\prime})$$

4) Digital Signature: ECDSA is used for signatures. Voter P_j generates his key pair (SigPK$_j$, SigSK$_j$) for signature.

5) Dishonest Peers: A peer is *dishonest* if he belongs to any of these three categories: (i) *Dishonest voter,* if he submits a ballot with invalid choice. (ii) *Dishonest reporter* if he does not report a ballot with invalid choice encrypted by his HEPK. (iii) *Dishonest counter* if he publishes incorrect tally result of those valid ballots encrypted by his HEPK.

C. The Voting Protocol

The proposed protocol consists of five stages and requires both offchain computations on blockchain client applications and onchain computations executed by smart contracts. The process flow of the voting protocol is shown in Figure 1.

Initially, voters (along with their two public keys [(P_j, HEPK$_j$, SigPK$_j$), $1 \leq j \leq n$]) are registered on the ledger with Status = "*honest*" and Vote Balance = 1.

1) Stage 1: Voting. During this stage, voters use client applications to prepare and submit their ballots on blockchain, within the voting time window. Each submitted voting transaction will be validated on blockchain and recorded on ledger if valid.

1.1) Voting—On Client by Peers

Algorithm 1.1—Vote(P_j, [B_k^j , $1 \leq k \leq K$])

```
for each ballot $B_k^j$
    generate the choice $C_k^j = N^{i-1} + r_k \times N^m$ by choosing an $i$ in $1 \leq i \leq m$ and picking a random number $r_k$.
    randomly pick an honest peer $P_{L_k^j}$ with $L_k^j \neq j$ and $P_{L_k^j}$ has not been picked by $P_j$.
    encrypt $C_k^j$, $EC_k^j$ = Encypt(HEPK$_{L_k^j}$, $C_k^j$).
sign and submit one voting transaction T with ($j$, [$B_k^j = (EC_k^j, L_k^j)$, $1 \leq k \leq K$], SigPK$_j$).
```

1.2) Validation—On Chain by Smart Contract

Algorithm 1.2—ValidateVote()

```
for each T with $B_1^j, \cdots, B_K^j$
    if $P_j$'s Vote Balance is 1
        if $L_1^j, \cdots, L_K^j$ are mutually different
            record T on ledger, change $P_j$'s Vote Balance to 0
            give $B_k^j$, $1 \leq k \leq K$, initial status "valid"
```

2) Stage 2: Two-Phase Verification of Each Ballot. During this stage, each ballot will be decrypted and verified by the peer whose HEPK was used to encrypt the ballot. Ballots with invalid choices should be reported to blockchain. These reported ballots will then be verified by smart contracts to check their validity. This helps to confirm whether a reported vote was actually dishonest (*dishonest voting*) or was incorrectly reported (*dishonest reporting*). In case a dishonest reporter is identified, all ballots encrypted by this reporter's HEPK need to be identified and replaced.

2.1) Verification—On Client by Peers

Algorithm 2.1—VerifyBallot(P_j)

```
for each $B_k^l = (EC_k^l, L_k^l)$ with $L_k^l = j$,
    decrypt $EC_k^l$, $\bar{C}_k^l$ = Decypt(HESK$_j$, $EC_k^l$)
    check if $\overline{C}_k^l$ is invalid
sign and submit one reporting transaction Report($P_j$) of all invalid ($B_k^l$, $\bar{C}_k^l$).
```

2.2) Verification—On Chain by Smart Contract

Algorithm 2.2—VerifyReport()

```
for each $P_j$, record to ledger (treat Report($P_j$) as empty if not received)
    for each ($B_k^l$, $\overline{C}_k^l$) in Report($P_j$)
        if $\overline{C}_k^l$ is invalid and $EC_k^l$ = Encypt(HEPK$_j$, $\bar{C}_k^l$)
            change the status of $P_l$ to "dishonest" and his ballots to "invalid" with timestamp
        else change the status of $P_j$ to "dishonest" and his ballots to "invalid" with timestamp
generate a list of these new dishonest peers
for each $P_l$ in list and each ballot $B$ encrypted by HEPK$_l$
    if Status($B$) = "valid", change it to "to be replaced"
```

Next proceed to Stage 3 if there is any ballot to be replaced; else proceed to Stage 4.

3) Stage 3: Revoting of Ballots Encrypted by Dishonest Peers' Public Key. For each ballot with status "to be replaced", a peer can either choose to give up replacement so that those ballots will simply be excluded from voting or choose to revote with a new ballot and encrypt it using another honest peer's HEPK. Then, all these replacement pairs can be broadcasted in one replacement transaction. The choice in this new ballot can be different from the original one[4].

3.1) Revote Ballots—On Client by Peers

Algorithm 3.1—RevoteBallot(P_j)

```
get the list of $P_j$'s ballots with status "to be replaced"
call Alg 1.1 Vote($P_j$, list) to generate new ballots
sign and submit one revoting transaction Revote($P_j$) with [(Original Ballot ID, New Ballot)]
```

3.2) Verify Revoting—On Chain by Smart Contract

Algorithm 3.2—VerifyRevoting()

```
for each $P_j$, record to ledger (treat Revote($P_j$) as empty if not received)
```

[4] This is for better privacy protection as this ballot has been once disclosed to a dishonest peer.

```
for each new ballot in Revote($P_j$), if the status of the ballot of the Original Ballot ID is "to be replaced"
    give this new ballot status "valid", change the original ballot's status to "replaced"
for the remaining ballots of $P_j$ with "to be replaced", change the status to "invalid"
```

Next proceed to Stage 4 if there is no new ballot in any revoting transaction, else repeat Stage 2 to verify the new ballots. Note that before the revoting period finishes, if any peer fails to cast replacing ballots, we consider that the peer give up replacements. Assuming that the number of ballots to be replaced is much smaller than K and sufficient time is given for revoting before timeout, this cut-out should have limited impacts on voting results.

4) Stage 4: Distributed Tally. During this stage, all valid ballots are tallied by peers whose status are still "honest" in a distributed way, namely each peer tallies his portion. Each individual tally result is published to blockchain. Then, smart contracts will verify each peer's tally result using the homomorphic encryption property. If there is any dishonest tally identified, the corresponding peer will be identified as dishonest, and Stage 3 will be repeated until no more dishonest tally is identified. To discourage malicious or intentional "drop-out" behavior in the tally stage, we follow a "no tally no voting" principle. This means that if a peer does not perform the tally assigned to him, then he will be marked as a dishonest peer and all his ballots will be discarded from counting. The underlying assumption is that an honest peer would report his tally result within sufficient time.

4.1) Distributed Tally—Client

```
Algorithm 4.1—DistributedTally($P_j$)
let $\mathrm{SUM}_j = 0$
for each valid $B_k^l = (EC_k^l, j)$, $\bar{C}_k^l = \mathrm{Decypt}(\mathrm{HESK}_j, EC_k^l)$
    $\mathrm{SUM}_j = \mathrm{SUM}_j + \bar{C}_k^l$
sign and submit a tally transaction Tally($P_j$) with $\mathrm{SUM}_j$
```

4.2) Verify Tally—On Chain by Smart Contract

```
Algorithm 4.2—VerifyTally()
for each $P_j$, record to ledger (treat Tally($P_j$) as empty if not received)
    count the number $t$ of valid ballots $B_{k_s}^{l_s} = (EC_{k_s}^{l_s}, j)$ encrypted by $\mathrm{HEPK}_j$
    compute $\beta_0(\mathrm{SUM}_j), \ldots, \beta_{m-1}(\mathrm{SUM}_j)$.
    if $\sum_{i=0}^{m-1} \beta_i(\mathrm{SUM}_j) \neq t$ or
            $EC_{k_1}^{j_1} \times \cdots \times EC_{k_t}^{j_t} \neq \mathrm{Encrypt}(\mathrm{HEPK}_j, \mathrm{SUM}_j)$
    change the status of $P_j$ to "dishonest" and the status of his ballots to "invalid" with timestamp
generate a list of these new dishonest peers
for each $P_l$ in list and each ballot $B$ encrypted by $\mathrm{HEPK}_l$
    if Status($B$) = "valid", change it to "to be replaced"
```

Next proceed to Stage 5 if there is no invalid tally and all honest peers have reported their tally results, otherwise repeat Stage 3.

5) Stage 5: Final Aggregation. Lastly, all individual tally results will be aggregated on chain by smart contracts and the voting result will be published on chain.

5.1) Final Aggregation—On Chain by Smart Contract

```
Algorithm 5.1—Aggregation()
aggregate distributed tally results
        $\mathrm{SUM} = \sum_{P_j \text{ is honest}} \mathrm{SUM}_j$
compute $\beta_0(\mathrm{SUM}), \ldots, \beta_{m-1}(\mathrm{SUM})$.
record to ledger SUM and the final voting result: the total no of ballots for option $i$ is $\beta_i(\mathrm{SUM})$, $0 \leq i < m$
```

This ends the voting protocol. Any peer can read the final voting result from the ledger and can also see and verify the history of events.

IV. Protocol Properties

In this section, we highlight some important characteristics of our protocol and explain how these ensure a fair voting.

A. Detectability of Cheating

We first prove how the proposed protocol can detect cheating without a trusted party. We can detect both simple individual/solo peer cheating and complicated collaborative cheating by a set of peers.

1) Solo Cheating: An invidual peer can behave dishonestly without colluding with other peers. In the context of the proposed protocol, solo cheating behaviors have been defined in Section III. It is straightforward to see from the protocol design that solo cheating behaviors can be directly detected either by an honest peer or by smart contracts.

2) Collaborative Cheating: This is a more complicated and insteresting scenario that involves at least two dishonest peers with certain cheating plan. In the proposed protocol, only steps 1.1, 2.1, 3.1 and 4.1 involve collaboration among peers, hence a meaningful collaborative cheating plan is: a dishonest peer, say Peer l, encrypts a ballot with invalid choice by another dishonest peer's HEPK, say HEPK$_j$, at Step 1.1 (or Step 3.1); Peer j decrypts this ballot at Step 2.1 and knows its invalidity, but does not report it and still counts it in its tally at Step 4.1. We prove below that this collaborative cheating will be detected. (Without loss of generality, we prove the two-peer case and the proof can be easily extended to multiple-peer case.)

Suppose the ballots encrypted by HEPK$_j$ and with status '*valid*' are

$$B_1 = (EC_1, j), \ldots, B_t = (EC_t, j)$$

and the tally result submitted by Peer j is SUM$_j$ at Stage 4. If all $C_1, \ldots, C_t$ are valid, then from the definition of a valid choice, we must have $\sum_{i=0}^{m-1} \beta_i(C_k) = 1$ for $1 \leq k \leq t$, thus

$$\sum_{i=0}^{m-1} \beta_i(C_1 + \cdots + C_t) = \sum_{i=0}^{m-1} \sum_{k=1}^{t} \beta_i(C_k) = \sum_{k=1}^{t} \sum_{i=0}^{m-1} \beta_i(C_k) = t$$

Hence if

$$\sum_{i=1}^{m} \beta_i(\mathrm{SUM}_j) \neq t,$$

then either $\mathrm{SUM}_j = C_1 + \cdots + C_t$ but at least one C_i is invalid, or $\mathrm{SUM}_j \neq C_1 + \cdots + C_t$.

If we verify that

$$EC_1 \times \cdots \times EC_t = \mathrm{Encrypt}(\mathrm{HEPK}_j, \mathrm{SUM}_j)$$

then we conclude $\mathrm{SUM}_j = C_1 + \cdots + C_t$and at least one C_i is invalid hence we can detect in Step 4.2 that Peer j did not report invalid ballots in Stage 2.

Otherwise SUM_j is not the correct sum of $C_1, \ldots, C_t$; namely we can conclude that Peer j did not tally $B_1, \ldots, B_t$ correctly.

To summarize, a peer needs to report in Stage 2 each invalid ballot encrypted by his HEPK and to submit the correct tally result in Stage 4; otherwise his cheating can be detected in Stage 4 and he will be excluded from both voting and tally as punishments.

B. *Correctability to Cheating*

In this section, we show how to correct detected cheating, without a trusted party, so that the voting result would not be impacted. To achieve this goal, we need to first exclude all dishonest peers' ballots. This can be done easily by labelling such ballots as "*invalid*" and exclude them in the tally stage. Second, we also need to exclude dishonest peers from participating in tally calculation. This is a trickier situation. In fact, the revoting stage (Stage 3) is precisely designed for this purpose. In Stage 3 we allow honest peers to revote their ballots previously encrypted by dishonest peers' HEPK, i.e. allow them to submit a replacement ballot encrypted by an honest peer's HEPK instead. By revoting, we can ensure that dishonest peers can no longer decrypt any of the valid ballots and hence are automatically excluded from tally. Thirdly, if a peer does not tally correctly, he will be detected at Step 4.2. All the ballots counted by the peer will be revoted, verified again and finally counted by honest peers.

At Step 2.2, 3.2 and 4.2, whenever there is a new dishonest peer detected, Step 3.1, 2.1 and 3.1 will be repeated, respectively, as shown in Figure 1. This iteration will ensure all dishonest peers are correctly and timely detected. One important note here is that there is no risk of forming an infinite loop as the number of dishonest peers are finite and no more than the total number of peers *n*.

C. *Correctness of the Voting Result*

By correctness of the voting result, we mean that the final tally result is the correct sum of all valid ballots, which implies 1) no invalid ballot is included in the final tally result; 2) all valid ballots are included and counted correctly. It is easy to see that the correctness of the proposed protocol follows directly from its detectability and correctability proved above. In addition, *integrity* and *auditability* are also guaranteed due to the fact that the voting data, such as voters' public keys, ballots, reports, tally results, status changes, etc., are all retrieved from blockchain ledger where these are recorded in a tamper-proof manner.

D. *Preserving Privacy of a Vote*

The privacy of a vote in our protocol is protected by "Distribution Voting" combined with encryption. In our design, each voter's vote is a distribution of his *K* ballots on the *m* options, and the *K* ballots are encrypted by *K* different randomly selected peers' public keys. In this way, each single ballot is disclosed to only the peer whose public key is used to encrypt it. Furthermore, even if a small group of peers collude, say *b peers*, at maximum they can only reveal a voter's *b* ballots. If *b* is less than ½ *K*, then that voter's preference is still safe from exposure. Therefore, for better privacy protection, *K* should not be small if the needed fault tolerance limit is high, i.e., *b* is big. More work is needed in the future to decide proper *K* for given *b*.

E. *Eligibility of Voters*

Our protocol assumes the voters are peers of a blockchain network. Since, a peer already has an identity on the network and there is non-negligible cost to become a peer, both in a permissioned blockchain and permission-less blockchain, the identities of peers can be used to ensure their eligibility as voters. Voter authentication is controlled by public key and signature associated with the identity of the peers. In addition, this protocol is also applicable to broader scenarios where the list of eligible voters is managed properly, for example, over a permissioned blockchain or within a group of users of a public blockchain with permission control in the group.

F. *Complexity*

Complexity could be a concern when the number of dishonest voters is relatively big, as certain part of detection and correction logic relies on smart contracts, in particular the verification process. In addition, whenever a new dishonest peer is discovered, voting, revoting and tally need to be repeated for affected ballots. At last, when a dishonest counter is discovered, his tally result is discarded so that the ballots he has counted need to be revoted and recounted by other peers. Also, his ballots need to be discarded, so the impacted honest counters need to exclude this counter's ballots and recount. There are possible methods to reduce the complexity, such as separating voters and counters. More work needs to be done in the future to address this.

V. Implementation

To verify the protocol, a reference prototype [31] was implemented on Hyperledger Fabric. Hyperledger Fabric is an open source blockchain platform targeting enterprise user scenarios. It enables building consortium blockchain networks with permission control and transaction consensus. Refer to [7] for the official documentation. For our prototype, we built an experimental network consisting of 20 peer nodes, each hosted in a docker container managed by a docker swarm cluster. The system configuration is illustrated in Table 1.

The reference implementation consists of two parts: client and smart contract. The communication flow among peers, smart contract and ledger are shown in Figure 2. Client holds the voting operations that will be performed by each individual voter. Smart contract maintains the voting logic that requires

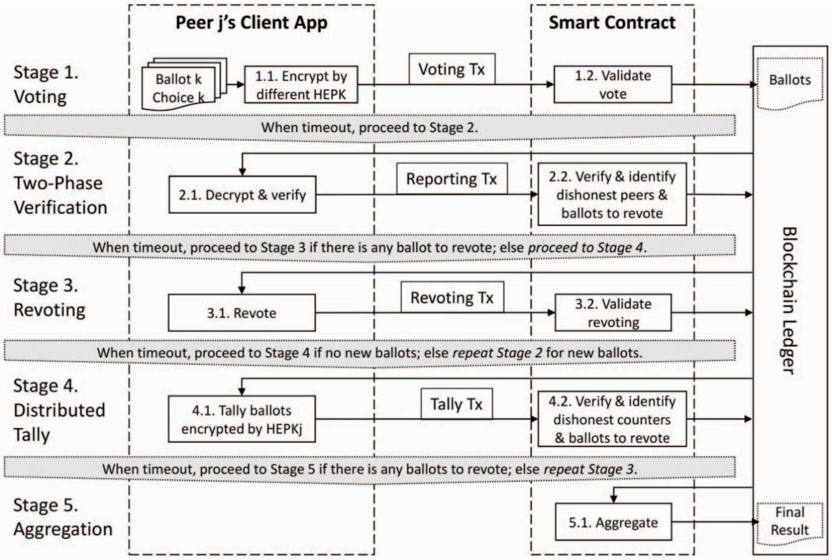
Peer j's Client App
Smart Contract
Stage 1.
Voting
Ballot k
Choice k
1.1. Encrypt by different HEPK
Voting Tx
1.2. Validate vote
Ballots
When timeout, proceed to Stage 2.
Stage 2.
Two-Phase
Verification
2.1. Decrypt & verify
Reporting Tx
2.2. Verify & identify dishonest peers & ballots to revote
When timeout, proceed to Stage 3 if there is any ballot to revote; else proceed to Stage 4.
Stage 3.
Revoting
3.1. Revote
Revoting Tx
3.2. Validate revoting
When timeout, proceed to Stage 4 if no new ballots; else repeat Stage 2 for new ballots.
Stage 4.
Distributed
Tally
4.1. Tally ballots encrypted by HEPKj
Tally Tx
4.2. Verify & identify dishonest counters & ballots to revote
When timeout, proceed to Stage 5 if there is any ballots to revote; else repeat Stage 3.
Stage 5.
Aggregation
5.1. Aggregate
Final Result
Blockchain Ledger

Figure 2. Overall communication flow among peers, smart contract and ledger of the voting protocol

collaboration and consensus between all voting participants. The detail of the implementation and main challenges addressed are elaborated further.

A. *Determinism of Smart Contract*

The first challenge to address is to ensure the determinism of smart contract. Ideally, we hope the entire voting flow can be driven by smart contract only to ensure that every step of voting, verifying and tally is done under the consensus of all voters. However, since the smart contract is executed in all peers within a network, its behavior must be deterministic to ensure different executions can always result in the same output. Hence in the reference implementation, all non-deterministic voting operations like key generation and selection of encryption key are included in the client side.

B. *Start and Completion of Voting Stages*

The second challenge to be addressed is to set start time and completion time of each stage. Because there is no global time in blockchain, to synchronize peers is a difficult problem. Nevertheless, our protocol is not sensitive to strict synchronization and we employ the following heuristics to solve the time issue.

Host of network	• Docker swarm cluster: 3 workers • Worker configuration: Linux VM with 8VCPU and 32GB memory
Fabric network topology	• Organization amount: 20 • Peer per organization: 1 • Orderer service running mode: solo • DB of ledger: Level-DB
Endorsement Policy	$\forall j \in [1, \ldots, 20]$
Version	• Fabric: 1.1.0-preview • Application/Client: NodeSDK-1.0.2

Table 1: Configuration of experiment system

To set the start time for each stage, we only need to set the global voting start time properly, since the protocol will execute automatically. We assume a voting promoter take the initiative to submit a "voting begins" message, which will be recorded on the blockchain. Peers who see this message on their ledger could start the voting.

For completion of each stage, we employ timeout window to synchronize all voters see Figure 2. In our implementation, the following method is applied to construct a distributed timer:

1) Each client initializes a timer locally with a global timeout setting, e.g. 5 minutes;
2) Client invokes a special transaction to report a timeout when its local timer expires;
3) Smart contract receives the transaction request and accumulate the counter of timeout report;
4) A special chaincode event "TIMEOUT" will be broadcasted to all clients if the threshold (e.g. 5) of the timeout counter is reached;
5) All clients receive the "TIMEOUT" event and get notified about the ending of specific voting stage.

C. *Workflow of Voting*

Following the protocol design in Section III, the workflow implemented consists of five stages as depicted in Figure 2, including voting, verification, revoting, tally and aggregation. In the following we describe how each stage is implemented.

1) Voting: Each client invokes a "vote" transaction with payload of K ballots [{uid, bid, ec, hepkid}, …, sig], where uid, bid, ec, hepkid and sig are user id, ballot id, encrypted choice, the HEPK id, signature, respectively. Smart contract validates each transaction and stores the ballots on ledger accordingly.

2) Verification: each client verifies those ballots encrypted by its HEPK and invokes a "report" transaction with payload of invalid ballots ([{bid, data}, …, sig]) if any. Smart contract verifies the signature and the validity of the report, and blacklists either the voter or the reporter correspondingly.

3) Revoting: Each client invokes a “revote” transaction with new ballots, each of which will replace a ballot encrypted using blacklisted hepkid. Smart contract checks if it is valid then invalidates the old ballot and stores the new one on ledger.

4) Tally: Each client invokes a “tally” transaction with the tally result of all the valid ballots encrypted by his public key. Smart contract verifies the signature and stores the tally result on ledger if it is correct or blacklists the counter otherwise.

5) Aggregation: Smart contract aggregates all the tally results and records the final voting result on ledger.

As illustrated in Figures 1 & 2, stage moving will be triggered by the TIMEOUT event and the “*Revoting*” stage could be executed iteratively in case a dishonest behavior is detected in the “*Verification*” or “*Tally*” stage.

We have conducted some initial experiment for small parameters with $n \leq 20$ voters, 2 options and small number of ballots for each vote and dishonest voters. The experiment result is presented on [31] but not here due to space limitation. Our experiment shows that our protocol is feasible and practical to implement on blockchain for a small scale of voters. In the future, more extensive testing will be conducted to evaluate the scalability and performance of the protocol.

VI. Conclusions and Future Work

In this paper we proposed a native voting mechanism on blockchain to facilitate decision making for peers of a blockchain network. The proposed protocol protects voters’ privacy and enables detection and correction against cheating without any trusted party. Our implementation on Hyperledger Fabric shows that the protocol is feasible and is practical for small to medium scale of voting problems.

More work needs to be done in the future. First, we need to perform formal security analysis. Potential security attacks, like cartel attack should be tested against the design. Second, a theoretical analysis of how to decide system parameters including number of ballots, PKI key length etc. should be conducted. Third, the current implementation [31] of our voting protocol is mainly for functional verification. More work on testing and optimization is needed in the future. We can also implement our protocol on different public and consortium blockchain platforms, e.g. Ethereum [2], Corda [3], Quorum [4], etc., to further inspect our protocol design, examine the performance difference between them.

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system.” www.bitcoin.org.

[2] Ethererum. www.ethereum.org.

[3] Corda. www.corda.net

[4] Quorum. www.jpmorgan.com/global/Quorum

[5] “DAO soft fork voting on Ethpool & Ethermine”, forum.ethereum.org/discussion/7796/dao-soft-fork-voting-on-ethpool-ethermine

[6] Carbonvote. carbonvote.com

[7] Hyperledger Fabric. www.hyperledger.org/projects/fabric

[8] Internet Policy Institue. “Report of the National Workshop on Internet Voting: Issues and Research Agenda “, 2001.

[9] A. Fujioka, T. Okamoto, and K. Ohta, “A practical secret voting scheme for large scale elections,” in Advances in Cryptology — AUSCRYPT ’92, Berlin, Heidelberg, 1993, pp. 244–251.

[10] J. D. Cohen and M. J. Fischer, “A robust and verifiable cryptographically secure election scheme,” 1985, pp. 372–382.

[11] J. C. Benaloh and M. Yung, “Distributing the power of a government to enhance the privacy of voters,” 1986, pp. 52–62.

[12] K. R. Iversen, “A Cryptographic Scheme for Computerized General Elections,” in Advances in Cryptology — CRYPTO ’91, 1992, pp. 405–419.

[13] H. Nurmi, A. Salomaa, and L. Santean, “Secret ballot elections in computer networks,” Comput. Secur., vol. 10, no. 6, pp. 553–560, Oct. 1991.

[14] D. Chaum, “Elections with Unconditionally-Secret Ballots and Disruption Equivalent to Breaking RSA,” in Advances in Cryptology — EUROCRYPT ’88, 1988, pp. 177–182.

[15] K. Sako, “Electronic voting scheme allowing open objection to the tally,” IEICE Trans. Fundam. Electron. Commun. Comput. Sci., vol. E77–A, no. 1, pp. 24–30, 1994.

[16] W.-S. Juang and C.-L. Lei, “A collision-free secret ballot protocol for computerized general elections,” Comput. Secur., vol. 15, no. 4, pp. 339–348, Jan. 1996.

[17] P. Feldman, “A practical scheme for non-interactive verifiable secret sharing,” 1987, pp. 427–438.

[18] J. Benaloh and D. Tuinstra, “Receipt-free secret-ballot elections (extended abstract),” 1994, pp. 544–553.

[19] B. Lee and K. Kim, “Receipt-Free Electronic Voting Scheme with a Tamper-Resistant Randomizer,” in Information Security and Cryptology — ICISC 2002, vol. 2587, P. J. Lee and C. H. Lim, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 389–406.

[20] R. Cramer, R. Gennaro, and B. Schoenmakers, “A Secure and Optimally Efficient Multi-Authority Election Scheme,” in Advances in Cryptology — EUROCRYPT ’97, vol. 1233, W. Fumy, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 103–118.

[21] B. Adida, “Helios: Web-based open-audit voting,” in 17th USENIX Security Symposium, San Jose, CA, USA, 2008.

[22] S. Estehghari and Y. Desmedt, “Exploiting the Client Vulnerabilities in Internet E-voting Systems: Hacking Helios 2.0 as an Example,” in 19th USENIX Security Symposium, Washington, DC, 2010.

[23] N. Chang-Fong and A. Essex, “The cloudier side of cryptographic end-to-end verifiable voting: a security analysis of Helios,” 2016, pp. 324–335.

[24] R. Wen and R. Buckland, “Masked Ballot Voting for Receipt-Free Online Elections,” in E-Voting and Identity, vol. 5767, P. Y. A. Ryan and B. Schoenmakers, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 18–36.

[25] N. Chondros et al., “Distributed, End-to-end Verifiable, and Privacy-Preserving Internet Voting Systems,” CoRR, vol. abs/1608.00849, 2016.

[26] D. Chaum and T. P. Pedersen, “Wallet Databases with Observers,” in Advances in Cryptology — CRYPTO’ 92, vol. 740, E. F. Brickell, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 89–105.

[27] D. L. Chung, “Distributed Helios: Defending Online Voting,” PhD Thesis, 2015.

[28] J. Bermúdez, “A practical multi-party computation algorithm for a secure distributed online voting system,” CoRR, vol. abs/1603.04228, 2016.

[29] K. Lee, J. James, T. Ejeta, and H. Kim, “Electronic Voting Service Using Block-Chain,” J. Digit. Forensics Secur. Law, 2016.

[30] C. Meter, “Design of Distributed Voting Systems,” CoRR, vol. abs/1702.02566, 2017.

[31] Reference Implementation of Peer Voting Protocol on Hyperledger Fabric. https://github.com/peer-voting