Proceedings Paper:
McCorry, Patrick, Shahandashti, Siamak F. orcid.org/0000-0002-5284-6847 and Hao, Feng (2017) A Smart Contract for Boardroom Voting with Maximum Voter Privacy. In: Kiayias, Aggelos, (ed.) Financial Cryptography and Data Security - 21st International Conference, FC 2017, Revised Selected Papers:21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers. Financial Cryptography and Data Security, 03-07 Apr 2017 Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) . Springer , MLT , pp. 357-375.

https://doi.org/10.1007/978-3-319-70972-7_20

A Smart Contract for Boardroom Voting with Maximum Voter Privacy

Patrick McCorry, Siamak F. Shahandashti and Feng Hao

School of Computing Science, Newcastle University UK
(patrick.mccorry, siamak.shahandashti, feng.hao)@ncl.ac.uk

Abstract. We present the first implementation of a decentralised and self-tallying internet voting protocol with maximum voter privacy using the Blockchain. The Open Vote Network is suitable for boardroom elections and is written as a smart contract for Ethereum. Unlike previously proposed Blockchain e-voting protocols, this is the first implementation that does not rely on any trusted authority to compute the tally or to protect the voter's privacy. Instead, the Open Vote Network is a self-tallying protocol, and each voter is in control of the privacy of their own vote such that it can only be breached by a full collusion involving all other voters. The execution of the protocol is enforced using the consensus mechanism that also secures the Ethereum blockchain. We tested the implementation on Ethereum's official test network to demonstrate its feasibility. Also, we provide a financial and computational breakdown of its execution cost.

1 Introduction

Ethereum is the second most popular cryptocurrency with a $870m market capitalisation as of November 2016. It relies on the same innovation behind Bitcoin [28]: namely, the Blockchain which is an append-only ledger. The Blockchain is maintained by a decentralised and open-membership peer-to-peer network. The purpose of the Blockchain was to remove the centralised role of banks for maintaining a financial ledger. Today, researchers are trying to re-use the Blockchain to solve further open problems such as coordinating the Internet of Things [20], carbon dating [6], and healthcare [10].

In this paper, we focus on decentralised internet voting using the Blockchain. E-voting protocols that support verifiability normally assume the existence of a public bulletin board that provides a consistent view to all voters. In practice, an example of implementing the public bulletin board can be seen in the yearly elections of the International Association of Cryptologic Research (IACR) [22]. They use the Helios voting system [1] whose bulletin board is implemented as a single web server. This server is trusted to provide a consistent view to all voters. Instead of such a trust assumption, we explore the feasibility of using the Blockchain as a public bulletin board. Furthermore, we consider a decentralised election setting in which the voters are responsible for coordinating the communication amongst themselves. Thus, we also examine the suitability

of the Blockchain's underlying peer-to-peer network as a potential authenticated broadcast channel.

There already exist proposals to use a Blockchain for e-voting. The Abu Dhabi Stock Exchange is launching a Blockchain voting service [19] and a recent report [3] by the Scientific Foresight Unit of the European Parliamentary Research Service discusses whether Blockchain-enabled e-voting will be a transformative or incremental development. In practice, companies such as The Blockchain Voting Machine [18], FollowMyVote [2] and TIVI [34] propose solutions that use *the Blockchain as a ballot box* to store the voting data.

These solutions achieve *voter privacy* with the involvement of a trusted authority. In FollowMyVote, the authority obfuscates the correspondence between the voter's real world identity and their voting key. Then, the voter casts their vote in plaintext. In TIVI, the authority is required to shuffle the encrypted votes before decrypting and counting the votes. In our work, we show that the voter's privacy does not need to rely on a central authority to decouple the voter's real world identity from their voting key, and the votes can be counted without the cooperation of a central authority. Furthermore, these solutions only use the Blockchain as an append-only and immutable global database to store the voting data. We propose that the network's consensus that secures the Blockchain can also enforce the execution of the voting protocol itself.

To date, both Bitcoin and Ethereum have inherent scalability issues. Bitcoin only supports a maximum of 7 transactions per second [8] and each transaction dedicates 80 bytes for storing arbitrary data. On the other hand, Ethereum explicitly measures computation and storage using a gas metric, and the network limits the gas that can be consumed by its users. As deployed today, these Blockchains cannot readily support storing the data or enforcing the voting protocol's execution for national scale elections. For this reason, we chose to perform **a feasibility study of a boardroom election** over the Blockchain which involves a small group of voters (i.e. 40 participants) whose identities are publicly known before the voting begins. For example, a boardroom election may involve stakeholders voting to appoint a new director.

We chose to implement the boardroom voting protocol as a smart contract on Ethereum. These smart contracts have an expressive programming language and the code is stored directly on the Blockchain. Most importantly, all peers in the underlying peer-to-peer network independently run the contract code to reach consensus on its output. This means that voters can potentially not perform all the computation to verify the correct execution of the protocol. Instead, the voter can trust the *consensus computing* provided by the Ethereum network to enforce the correct execution of the protocol. This enforcement turns *detection* measures seen in publicly verifiable voting protocols into *prevention* measures.

Our contributions. We provide the first implementation of a decentralised and self-tallying internet voting protocol. The Open Vote Network [17] is a boardroom scale voting protocol that is implemented as a smart contract in Ethereum. The Open Vote Network provides maximum voter privacy as an individual vote can only be revealed by a full-collusion attack that involves compromising all

other voters; all voting data is publicly available; and the protocol allows the tally to be computed without requiring a tallying authority. Most importantly, our implementation demonstrates the feasibility of using the Blockchain for decentralised and secure e-voting.

2 Background

2.1 Self-Tallying Voting Protocols

Typically, an e-voting protocol that protects the voter's privacy relies on the role of a trustworthy authority to decrypt and tally the votes in a verifiable manner. E-voting protocols in the literature normally distribute this trust among multiple tallying authorities using threshold cryptography; for example, see Helios [1]. However, voters still need to trust that the tallying authorities do not collude altogether, as in that case, the voter's privacy will be trivially breached.

Remarkably, Kiayias and Yung [24] first introduced a *self-tallying* voting protocol for boardroom voting with subsequent proposals by Groth [16] and Hao et al. [17]. A self-tallying protocol converts tallying into an open procedure that allows any voter or a third-party observer to perform the tally computation once all ballots are cast. This removes the role of a tallying authority in an election as anyone can compute the tally without assistance. These protocols provide *maximum ballot secrecy* as a full collusion of the remaining voters is required to reveal an individual vote and *dispute-freeness* that allows any third party to check whether a voter has followed the voting protocol correctly. Unfortunately, self-tallying protocols have a fairness drawback as the last voter can compute the tally before anyone else[1] which results in both *adaptive* and *abortive* issues.

The adaptive issue is that knowledge of the tally can potentially influence how the last voter casts their vote. Kiayias and Yung [24] and Groth [16] propose that an election authority can cast the final vote which is excluded from the tally. However, while this approach is applicable to our implementation discussed later, it effectively re-introduces an authority that is trusted to co-operate and not to collude with the last voter. Instead, we implement an optional round that requires all voters to hash their encrypted vote and store it in the Blockchain as a commitment. As a result, the final voter can still compute the tally, but is unable to change their vote.

The abortive issue is that if the final voter is dissatisfied with the tally, they can abort without casting their vote. This abortion prevents all other voters and third parties from computing the final tally. Previously, Kiayias and Yung [24] and Khader et al. [23] proposed to correct the effect of abortive voters by engaging the rest of the voters in an additional recovery round. However, the recovery round requires full cooperation of all the remaining voters, and will fail if any member drops out half-way. We highlight that the Blockchain and

[1] It is also possible for voters that have not yet cast their vote to collude and compute the partial tally of the cast votes. For simplicity, we discuss a single voter in regards to the fairness issue.

smart contracts can enforce a financial incentive for voter participation using a deposit and refund paradigm [25]. This allows providing a new countermeasure to address the abortive issue: all voters deposit money into a smart contract to register for an election and are refunded upon casting their vote. Any voter who does not vote before the voting deadline simply loses their deposit.

In the next section we present Open Vote Network [17] before discussing its smart contract implementation on Ethereum. We chose to implement this protocol instead of others (e.g., [16,24]) because it is the most efficient boardroom voting protocol in the literature in each of the following aspects: the number of rounds, the computation load per voter and the bandwidth usage [17]. As we will detail in Section 3, the efficiency of the voting protocol is critical as even with the choice of the most efficient boardroom voting protocol, its implementation for a small-scale election is already nearing the capacity limit of an *existing* Ethereum block.

2.2 The Open Vote Network Protocol

The Open Vote Network is a decentralized two-round protocol designed for supporting small-scale boardroom voting. In the first round, all voters register their intention to vote in the election, and in the second round, all voters cast their vote. The systems assumes an authenticated broadcast channel is available to all voters. The self-tallying property allows anyone (including non-voters) to compute the tally after observing messages from the other voters. In this paper, we only consider an election with two options, e.g., yes/no. Extending to multiple voting options, and a security proof of the protocol can be found in [17].

A description of the Open Vote Network is as follows. First, all n voters agree on (G, g) where G denotes a finite cyclic group of prime order q in which the Decisional Diffie-Hellman (DDH) problem is intractable, and g is a generator in G. A list of eligible voters $(P_1, P_2, ..., P_n)$ is established and each eligible voter P_i selects a random value $x_i \in_R \mathbb{Z}_q$ as their private voting key.

Round 1. Every voter P_i broadcasts their voting key g^{x_i} and a (non-interactive) zero knowledge proof $ZKP(x_i)$ to prove knowledge of the exponent x_i on the public bulletin board. $ZKP(x_i)$ is implemented as a Schnorr proof [32] made non-interactive using the Fiat-Shamir heuristic [15].

At the end, all voters check the validity of all zero knowledge proofs before computing a list of reconstructed keys:

$$Y_i = \prod_{j=1}^{i-1} g^{x_j} / \prod_{j=i+1}^{n} g^{x_j}$$

Implicitly setting $Y_i = g^{y_i}$, the above calculation ensures that $\sum_i x_i y_i = 0$.

Round 2. Every voter broadcasts $g^{x_i y_i} g^{v_i}$ and a (non-interactive) zero knowledge proof to prove that v_i is either no or yes (with respect to 0 or 1) vote. This one-out-of-two zero knowledge proof is implemented using the Cramer, Damgård and Schoenmakers (CDS) technique [7].

All zero knowledge proofs must be verified before computing the tally to ensure the encrypted votes are well-formed. Once the final vote has been cast, then anyone (including non-voters) can compute $\prod_i g^{x_i y_i} g^{v_i}$ and calculate $g^{\sum_i v_i}$ since $\prod_i g^{x_i y_i} = 1$ (see [17]). The discrete logarithm of $g^{\sum_i v_i}$ is bounded by the number of voters and is a relatively small value. Hence the tally of yes votes can be calculated subsequently by exhaustive search.

Note that for the election tally to be computable, all the voters who have broadcast their voting key in Round 1 must broadcast their encrypted vote in Round 2. Also note that in Round 2, the last voter to publish their encrypted vote has the ability to compute the tally before broadcasting their encrypted vote (by simulating that he would send a no-vote). Depending on the computed tally, he may change his vote choice. In our implementation, we address this issue by requiring all voters to commit to their votes before revealing them, which adds another round of commitment to the protocol.

The decentralised nature of the Open Vote Network makes it suitable to implement over a Blockchain. Bitcoin's blockchain could be used as the public bulletin board to store the voting data for the Open Vote Network. However, this requires the voting protocol to be externally enforced by the voters. Instead, we propose using Ethereum to enforce the voting protocol's execution. This is possible as conceptually Ethereum can be seen as a global computer that can store and execute programs. These programs are written as smart contracts, and their correct execution is enforced using the same network consensus that secures the Ethereum blockchain. Furthermore, its underlying peer-to-peer network can act as an authenticated broadcast channel.

2.3 Ethereum

In this section, we focus on the types of accounts available, the transaction structure and the Blockchain protocol used in Ethereum.

Ethereum has two account types:

- An **externally owned account (user-controlled)** is a public-private key pair controlled by the user. We denote these accounts by $A, B, \ldots$.
- A **contract account** is a smart contract that is controlled by its code. We denote a smart contract account by λ.

Both account types can store the Ethereum currency 'ether'. Ethereum does not perform computation in a contract without user interaction. As such, a contract account must be activated by a user-controlled account before its code can be executed. Executing code requires the user-controlled account to purchase 'gas' using the ether currency and a gas price set by the user determines the conversion rate of ether to gas. The cost of gas is essentially a transaction fee to encourage miners to include the code execution in the Blockchain. Most importantly, gas is a metric that standardises the cost of executing code on the network and each assembly operation (opcode) has a fixed gas cost based on its expected execution time.

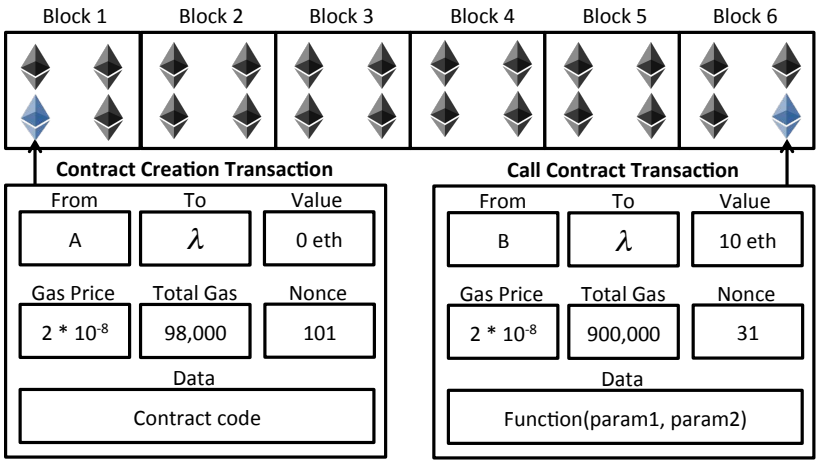

Fig. 1. Alice creates a smart contract on the Blockchain and the contract code is sent in the transaction's 'data' field. The contract is given an address λ. Bob can call a function of the contract using a second transaction sending gas to the address λ.

An **Ethereum Transaction**'s structure can be seen in Figure 1. Each field of the transaction is described below:

- **From:** A signature from a user-controlled account to authorise the transaction.
- **To:** The receiver of the transaction and can be either a user-controlled or contract address.
- **Data:** Contains the contract code to create a new contract or execution instructions for the contract.
- **Gas Price:** The conversion rate of purchasing gas using the ether currency.
- **Total Gas:** The maximum amount of gas that can be consumed by the transaction.
- **Nonce:** A counter that is incremented for each new transaction from an account.

The Ethereum blockchain is considered an orderly transaction-based state machine. If multiple transactions call the same contract, then the contract's final state is determined by the order of transactions that are stored in the block. Strictly, Ethereum's blockchain is a variation of the GHOST protocol [33] which is a tree-based blockchain. This tree has a main branch of blocks that is represents the 'Blockchain' and transactions in these blocks determine the final state of contracts and account balances. Similar to Bitcoin, the security of the Blockchain relies upon miners providing a 'proof of work' which authorises the miner to append a new block. This proof of work is a computationally difficult puzzle, and the miner is rewarded 5 ether if the block is successfully appended.

Blocks are created approximately every twelve seconds in Ethereum which is significantly faster than Bitcoin's 10 minutes interval. As a consequence, it

is probabilistically more likely that two or more blocks are created by different miners at the same time. In Bitcoin, only one of the competing blocks can be accepted into the Blockchain, and the remaining blocks are discarded. However, in Ethereum, these discarded blocks are appended to the Blockchain as leaf nodes ('uncle blocks'). It should be noted that the uncle block still rewards the miner a proportion of the 5 ether block reward based on when the block is included in the Blockchain.

Ethereum's blockchain provides a natural platform for the Open Vote Network. It provides a public bulletin board and an authenticated broadcast channel which are necessary in decentralised internet voting protocols to support coordination amongst voters. As well, almost all computations in the Open Vote Network are public computations that can be written as a smart contract. Most importantly, the entire execution of the voting protocol is enforced by the same consensus that secures the Blockchain. In the next section, we discuss our implementation and the feasibility of performing internet voting over the Blockchain.

3 The Open Vote Network over Ethereum

We present an implementation of the Open Vote Network over Ethereum. The code is publicly available[2]. Three HTML5/JavaScript pages are developed to provide a browser interface for all voter interactions. The web browser interacts with an Ethereum daemon running in the background. The protocol is executed in five stages, and requires voter interaction in two (and an optional third) rounds. We give an overview of the implementation in the following.

3.1 Structure of Implementation

There are two smart contracts that are both written in Ethereum's Solidity language. The first contract is called the voting contract. It implements the voting protocol, controls the election process and verifies the two types of zero knowledge proofs we have in the Open Vote Network. The second contract is called the cryptography contract. It distributes the code for creating the two types of zero knowledge proofs[3]. This provides all voters with the same cryptography code that can be used locally without interacting with the Ethereum network. We have also provided three HTML5/JavaScript pages for the users:

- **Election administrator** (`admin.html`) administers the election. This includes establishing the list of eligible voters, setting the election question, and activating a list of timers to ensure the election progresses in a timely manner. The latter includes notifying Ethereum to begin registration, to close registration and begin the election, and to close voting and compute the tally.

[2] https://(web address hidden for submission)

[3] We have included the code to create and verify the two types of zero knowledge proofs in the cryptography contract. The code is independent of the Open Vote Network and can be used by other smart contracts.

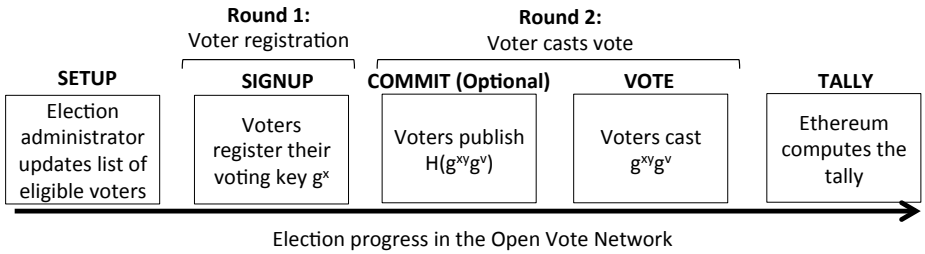

Fig. 2. There are five stages to the election.

- **Voter** (`vote.html`) can register for an election, and once registered must cast their vote.
- **Observer** (`livefeed.html`) can watch the election's progress consisting of the election administrator starting and closing each stage and voters registering and casting votes. The running tally is not computable.

We assume that voters and the election administrator have their own Ethereum accounts. The Web3 framework is provided by the Ethereum Foundation to facilitiate communication between a user's web browser and their Ethereum client. The user can unlock their Ethereum account (decrypt their Ethereum's private key using a password) and authorise transactions directly from the web browser. There is no need for the user to interact with an Ethereum wallet, and the Ethereum client can run in the background as a daemon.

3.2 Election stages

Figure 2 presents the five stages of the election in our implementation. The smart contract has a designated owner that represents the election administrator. This administrator is responsible for authenticating the voters with their user-controlled account and updating the list of eligible voters. A list of timers is enforced by the smart contract to ensure that the election progresses in a timely manner. The contract only allows eligible voters to register for an election, and registered voters to cast a vote. Furthermore, the contract can require each voter to deposit ether upon registration, and automatically refund the ether when their vote is accepted into the Blockchain. Each stage of the election is described in more detail below:

SETUP. The election administrator authenticates each voter with their user-controlled account and updates the voting contract to include a whitelist of accounts as eligible voters. He defines a list of timers to ensure that the election progresses in a timely manner:

- $t_{finishRegistration}$: all voters must register their voting key g^{x_i} before this time.
- $t_{beginElection}$: the election administrator must notify Ethereum to begin the election by this time.

- $t_{finishCommit}$: all voters must commit to their vote $H(g^{x_i y_i} g^{v_i})$ before this time. This is only used if the optional COMMIT stage is enabled.
- $t_{finishVote}$: all voters must cast their vote $g^{x_i y_i} g^{v_i}$ before this time.
- π: a minimum length of time in which the commitment and voting stages must remain active to give voters sufficient time to vote.

The administrator also sets the registration deposit d, the voting question, and if the optional COMMIT stage should be enabled. Finally, the administrator notifies Ethereum to transition from the SETUP to the SIGNUP stage.

SIGNUP. All eligible voters can choose to register for the vote after reviewing the voting question and other parameters set by the election administrator. To register, the voter computes their voting key g^{x_i} and $ZKP(x_i)$. Both the key and proof are sent to Ethereum alongside a deposit of d ether. Ethereum does not accept any registrations after $t_{finishRegistration}$. The election administrator is responsible for notifying Ethereum to transition from the SIGNUP to either the optional COMMIT or the VOTE stage. All voter's reconstructed keys $g^{y_0}, g^{y_1}, ..., g^{y_n}$ are computed by Ethereum during the transition.

COMMIT(Optional). All voters publish a hash of their vote $H(g^{x_i y_i} g^{v_i})$ to the Ethereum blockchain. The contract automatically transitions to the VOTE stage once the final commitment is accepted into the Blockchain.

VOTE. All voters publish their (encrypted) vote $g^{x_i y_i} g^{v_i}$ and a one-out-of-two zero knowledge proof to prove that v_i is either zero or one. The deposit d is refunded to the voter when their vote is accepted by Ethereum. The election administrator notifies Ethereum to compute the tally once the final vote is cast.

TALLY. The election administrator notifies Ethereum to compute the tally. Ethereum computes the product of all votes $\prod_i g^{x_i y_i} g^{v_i} = g^{\sum_i v_i}$ and brute forces the discrete logarithm of the result to find the number of yes votes.

As mentioned before, Open Vote Network requires all the registered voters to cast a vote to enable tally calculation. The deposit d in our implementation provides a financial incentive for registered voters to vote. This deposit is returned to the voter if they follow through with the voting protocol and do not drop out. The list of timestamps defined by the election administrator determines if the voter's deposit d is forfeited or refunded. There are three refund scenarios if a deadline is missed:

- Registered voters can claim their refund if the election does not begin by $t_{beginElection}$.
- Registered voters who have committed can claim their refund if not all registered voters commit to their vote by $t_{finishCommit}$.
- Registered voters can claim their refund if not all registered voters cast their vote by $t_{finishVote}$.

4 Design Choices

In this section, we discuss the design choices we made when developing the implementation. In particular, we elaborate on some attack vectors that are addressed

in our smart contract and clarify the trust assumptions that are required for our implementation to be secure.

Individual and public verifiability. We assume that the voter's machine, including their web browser, is not compromised. The voter has an incentive to ensure their machine is secure. If the machine or web browser is compromised, the voter's ether is likely to be stolen. The voter can check that their vote has been *recorded as cast* and *cast as intended* by inspecting the Blockchain and decrypting their vote using the key x_i. Also, the voter, or any observer for that matter, can independently compute the tally to verify that the cast votes have been *tallied as recorded*. Unfortunately, this public verifiability does not provide any coercion resistance as the voting is conducted in a "unsupervised" environment. The voter may vote under the direct duress of a coercer who stands over their shoulder. The voter can also reveal x to prove how their vote was cast to others. As such, in a similar fashion to Helios [1], we note that our implementation is only suitable for low-coercion elections.

Voter authentication. Smart contracts can call other smart contracts. As a result, there exist two methods to identify the caller. The first is `tx.origin` that identifies the user-controlled account that authorised the transaction, and not the immediate caller. The second is `msg.sender` that identifies the immediate caller which can be a contract or a user-controlled address. Initially, a developer might use `tx.origin` as it appears the approppriate choice to identify the voter. Unfortunately, this approach allows a malicious smart contract to impersonate the voter and register for an election.

To illustrate, a voter is given the interface to a smart contract called BettingGame. This lets the voter place a bet using `BettingGame.placeBid()`. Unknowingly to the voter, if this function is called, then BettingGame will call `TheOpenVoteNetwork.register()` and register a voting key on behalf of the voter. To overcome this issue, we recommend using `msg.sender` as it identifies the immediate caller whose address should be in the list of eligible voters.

Defending against replay attacks. All voting keys g^{x_i} and their zero knowledge proofs $ZKP(x_i)$ are publicly sent to the Ethereum blockchain. A potential attack is that another eligible voter can attempt to register the same voting keys by replaying g^{x_i} and $ZKP(x_i)$. This would also let them later copy the targeted voter's vote. We highlight that the commitment (i.e., input arguments to the hash function) in the zero knowledge proof includes `msg.sender` and Ethereum will not accept the zero knowledge proof $ZKP(x_i)$ if `msg.sender` does not match the account that is calling the contract. As such, it is not possible to replay another voter's key g^{x_i} without their co-operation. This also applies to the one-out-of-two zero knowledge proofs.

Blocking re-entrancy. A hacker recently exploited a re-entrancy vulnerability in theDAO to steal over 3.6 million ether. Luu et al highlight [26] that 186 distinct smart contracts stored on the Blockchain (including theDAO) are also potentially vulnerable. This attack relies on the contract sending ether to the user before deducting their balance. The attacker can recursively call the contract in such a way that the sending of ether is repeated, but the balance is only

deducted once. To prevent this attack, we follow the advice of Reitwiessner [30] to first deduct the voter's balance before attempting to send the ether.

The role of timers. The election administrator sets a list of timers to allow Ethereum to enforce that the election progresses in a timely manner. A minimum time interval π (unit in seconds) is set during the SETUP stage to ensure each stage remains active for at least a time interval of length π. In particular, the rules $t_{finishCommit} - t_{beginElection} > \pi$ and $t_{finishVote} - t_{finishCommit} > \pi$ are enforced to provide sufficient time for voters to commit to and cast their vote. Also, it provides a window for the voter's transaction to be accepted into the Blockchain. This is necessary to prevent a cartel of miners ($< 51\%$) attempting to censor some transactions. Voters need to check that π is not a small value such as $\pi = 1$. In this case, the voting stage can finish one second after the election begins. As a result, all voters are likely to lose their deposits. Of course, both the COMMIT and VOTE stage can finish early if all voters have participated.

The block's timestamp is used to enforce the above timers. Ethereum has a tight bound on the timestamp which must conform to the following two rules. First, a new block's timestamp must be greater than the previous block. Second, the block's timestamp must be less than the user's local clock. Furthermore, the miner's ability to drift a block's timestamp by 900 seconds (15 minutes) as reported in [26] is no longer possible [11].

Ethereum miners. The tip of the Blockchain is uncertain and the state of a contract at the time of signing a transaction is not guaranteed to remain the same. Furthermore, miners control the order of transactions in a block, and can control the order of a contract's execution if there are two or more transactions calling the same contract. Although the order of voting keys or casting a vote does not matter, the order of transactions is important if a timer is about to expire. For example, if the voter attempts to register around the time that $t_{finishRegistration}$ expires, then the miner can prevent the registration in two ways. First, the miner can choose a block timestamp that expires the $t_{finishRegistration}$ timer. Second, if the miner has the voter's registration transaction and the election administrator's begin election transaction, he can order the transactions in the block such that the smart contract begins the election before allowing the voter to register for the election. Unfortunately, in both cases, the voter's registration will fail.

It is important that voters authorise their transactions in good time before the stage is destined to end. We must assume that the majority of miners are not attempting to disrupt the election. A smaller cartel of miners ($< 51\%$) can potentially delay transactions being accepted into the Blockchain using techniques such as selfish mining [14] [31] or feather forking [29]. This ability of miners to delay a transaction is a fundamental problem for any contract.

The election administrator. An election administrator is required to add voters to the list of eligible voters, set the election's parameters and to begin the registration stage. Unfortunately, smart contracts cannot execute code without the notification of an external user-controlled account. As such, a user is still required to notify the smart contract to begin the election and compute the

tally. Deciding who is responsible for notifying Ethereum is an implementation trade-off and we have assumed it is the election administrator's role. If necessary, the contract can be modified to allow any registered voter to perform the notification. However, in that case it is possible that two or more voters attempt to notify Ethereum at the same time and broadcast transactions to the network. If this happens, only one transaction can begin the election or compute the tally. All unsuccessful transactions will still be stored in the Blockchain and all the broadcasting users will still be charged transaction fees.

Removing the COMMIT stage. The COMMIT stage prevents the final voter computing the tally and using this information to decide how to vote. It is possible to remove this stage if we require the election administrator (or a separate external party) to perform some extra tasks. In this case, the administrator is the first voter to register a voting key g^x and deposit of d ether before voter registration begins. Next, he is required to merely reveal his secret x once all voters have cast their vote. Revealing x allows Ethereum to calculate a final dummy vote and compute the tally. The administrator is now trusted not to collude with the last voter. This approach removes the COMMIT phase but requires extra an trust assumption.

Do voters need to use Ethereum? Today, all voters need to download the full Ethereum blockchain to confirm the voting protocol is being executed correctly. In the future, voters will be able to use the Light Ethereum Subprotocol (LES) [12] which is similar to Bitcoin's simplified payment verification (SPV) protocol. In LES, voters will only verify the voting protocol's state and not be required to store the full Blockchain.

Most importantly, it is possible for the voter to participate in the voting protocol without the full Blockchain. In this case, the voter merely broadcasts their transactions and trusts the consensus mechanism of the Ethereum network to enforce the correct execution of the protocol. This would enable voters who have devices with limited resources to vote in our implementation. We have provided `livefeed.html` to allow voters to visit an external website and confirm their registration or vote has been recorded in the Blockchain.

5 Experiment on Ethereum's Test Network

Our implementation was deployed on Ethereum's official test network that mimics the production network. We sent 126 transactions to simulate forty voters participating in the protocol. Each transaction's computational and financial cost is outlined in Table 1. Each transaction by the election administrator (denoted by the prefix 'A:' in the table) is broadcast only once, and each transaction by a voter (denoted by the prefix 'V:' in the table) is broadcast once per voter, i.e., a total of 40 times. Also, we have rounded the cost in US Dollars (denoted by $) to two decimal places.

We had to split the Open Vote Network into two contracts as the code was too large to store in an Ethereum block which has a capacity of approximately 4.7 million gas. The voting contract VoteCon (80% of block capacity, and $0.83

Entity: Transaction	Cost in Gas	Cost in $
A: VoteCon	3,779,963	0.83
A: CryptoCon	2,435,848	0.54
A: Eligible	2,153,461	0.47
A: Begin Signup	234,984	0.05
V: Register	763,118	0.17
A: Begin Election	3,085,449	0.68
V: Commit	70,112	0.02
V: Vote	2,490,412	0.55
A: Tally	746,485	0.16
Administrator Total	12,436,190	2.74
Voter Total	3,323,642	0.73
Election Total	145,381,858	31.98

Table 1. A breakdown of the costs for 40 participants using the Open Vote Network. We have approximated the cost in USD ($) using the conversion rate of 1 ether = $11 and the gas price of 0.00000002 ether which are the real world costs in November 2016. Also, we have identified the cost for the election administrator 'A' and the voter 'V'.

transaction fee) contains the protocol logic. The cryptography contract CryptoCon (52% of block capacity, and $0.54 transaction fee) contains the code to create and verify the two types of zero knowledge proofs we have in the protocol.

CryptoCon can be reused by other contracts requiring similar zero knowledge proofs. It is important to note that the code for computing the zero knowledge proofs is run locally on the voter's machine, and no transactions are sent to the network. CryptoCon's purpose is to ensure that all voters have access to the same cryptography code.

As the figures show, voter registrations and vote casting cost around 16% and 53% of block capacity, respectively. This suggests that the current block sizes in Ethereum support at most six voter registration per block and at most one vote casting per block. Recall that blocks are currently generated in Ethereum at a rate of one block per 12 seconds.

Overall, running the election with 40 voters costs the election administrator $2.74. The total election cost including the cost for the administrator and the voters is $31.98 which breaks down to a reasonable cost of $0.73 per voter.

To see how the cost for the election administrator and the voter vary with different number of voters we have carried out experiments with 5, 10, 15, ..., 60 voters. Figure 3 highlights the distribution of cost for the election administrator and the voter based on the number of voters participating in the election. This shows that the election administrator's cost increases linearly based on the number of voters, and the voter's cost remains constant.

All testing was performed on the test network due to an ongoing DoS attack, starting from 22 September 2016, on Ethereum's production network [5]. Miners set the block's gas limit to 1,500,000 gas to reduce the impact on the network and a hard fork [4] was deployed on 18 October 2016 to prevent the attack. How-

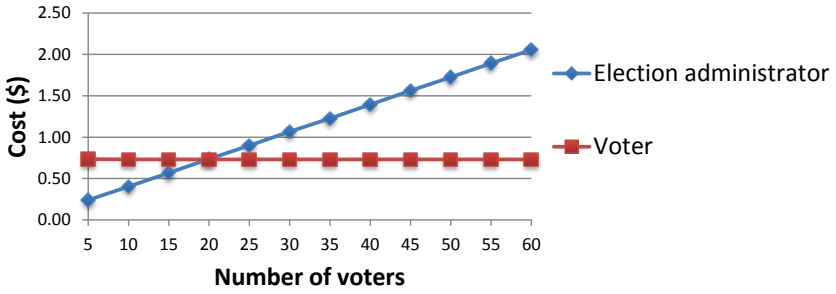

Fig. 3. The average cost for the election administrator and the voter based on the number of voters participating in the election.

ever, a second DoS attack began on 19 October 2016. Ethereum developers have recommended a temporary gas limit of 2,000,000 until the next scheduled hard fork. As such, the Open Vote Network cannot run on the production network at this time.

5.1 Timing Analysis

Table 2 outlines the timing analysis measurements for tasks in the Open Vote Network. All measurements were performed on a MacBook Pro running OS X 10.10.5 equipped with 4 cores, 2.3GHz Intel Core i7 and 16 GB DDR3 RAM. All time measurements are rounded up to the next whole millisecond. We use the Web3 framework to facilitate communication between the web browser and the Ethereum daemon. All tasks are executed using `.call()` that allows us to measure the code's computation time on the local daemon.

The cryptography smart contract is responsible for creating the zero knowledge proofs for the voter. The time required to create the proofs is 81 ms for the Schnorr proof and 461 ms for the one-out-of-two zero knowledge proof. These actions are always executed using `.call()` as this contract should never receive transactions.

The voting smart contract is responsible for enforcing the election process. Registering a vote involves verifying the Schnorr zero knowledge proof and in total requires 142 ms. To begin the election requires computing the reconstructed public keys which takes 277 ms in total for forty voters. Casting a vote involves verifying the one-out-of-two zero knowledge proof which requires 573 ms. Tallying involves summing all cast votes and brute-forcing the discrete logarithm of the result and on average takes around 132 ms.

We decided to distribute the cryptography code using the Ethereum blockchain to allow all voters to use the same code. Running the code on the voter's local daemon is significantly slower than using a seperate library such as OpenSSL. For example, creating a Schnorr signature using OpenSSL on a comparable machine requires 0.69 ms [27]. This slowness is mostly due to the lack of native support for elliptic curve math in Ethereum smart contracts. The Ethereum Foundation

Action	Avg. Time (ms)
Create ZKP(x)	81
Register voting key	142
Begin election	277
Create 1-out-of-2 ZKP	461
Cast vote	573
Tally	132

Table 2. A time analysis for actions that run on the Ethereum daemon.

has plans to include native support and we expect this to significantly improve our reported times.

6 Discussion on Technical Difficulties

In this section, we discuss the difficulties faced while implementing the Open Vote Network on Ethereum.

Lack of support for cryptography. Ethereum supports up to 256-bit unsigned integers. For this reason, we chose to implement the protocol over an elliptic curve instead of a finite field. However, Solidity does not currently support Elliptic Curve cryptography, and we had to include an external library to perform the computation. Including the library led to our voting contract becoming too large to store on the Blockchain. As previously discussed, we separated the program into two smart contracts: one voting contract and one cryptography contract. The cryptographic computations are expensive and this results in a block only being able to support six voter registrations, or a single vote.

Call stack issues. The call stack of a program has a hard-coded limit of 1024 stack frames. This limits the amount of local memory available, and the number of function calls allowed. These limitations led to difficulty while implementing the 1-out-of-2 ZKP as the temporary variables typically required exceeded the hard-coded limit of 16 local variables [21]. We had to use variables extremely sparingly to ensure that the 1-out-of-2 ZKP could be implemented.

Lack of debugging tools. The Mix IDE that provides a solidity source code debugger has been discountined [13] and could not be used for our work. Remix is the replacement for the Mix IDE and it provides a debugger for contracts at the assembly level, but this is too low for debugging Solidity contracts. Instead, we had to create `Events` that log data along with the contract to help with debugging which is incorporated into the contract before deployment.

Mitigate loss of voting key. The voting key is kept secret by the voter and needs to be stored on their local machine. This is important to ensure that if the voter's web browser crashes or is closed, then the voting key is not lost. We provide a standalone Java program `votingcodes.jar` to generate the voting key and store it in `votingcodes.txt`. The voter is required to upload this file to their web browser.

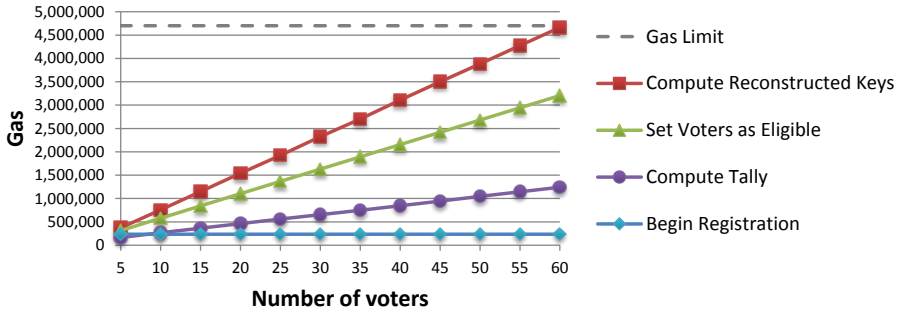

Fig. 4. The gas cost for the election administrator based on the number of voters participating in the election.

Maximum number of voters. Figure 4 demonstrates the results of our experiment and highlights the breakdown of the election administrator's gas consumption. Except for opening registration, the gas cost for each task increases linearly with the number of voters. The gas limit for a block was set at 4.7 million gas by the miners before the recent DoS attacks. This means that the smart contract reaches the computation and storage limit if it is computing the voter's reconstructed keys for around sixty registered voters. This limit exists as all keys are computed in a single transaction and the gas used must be less than the block's gas limit. To avoid reaching this block limit, we currently recommend a safe upper limit of around 50 voters. However, the contract can be modified to perform the processing in batches and allow multiple transactions to complete the task.

7 Conclusion

In this paper, we have presented a smart contract implementation for the Open Vote Network that runs on Ethereum. Our implementation was tested on the official Ethereum test network with forty simulated voters. We have shown that our implementation can be readily used with minimal setup for elections at a cost of $0.73 per voter. The cost can be considered reasonable as this voting protocol provides maximum voter privacy and is publicly verifiable. This is the first implementation of a decentralised internet voting protocol running on a Blockchain. It uses the Ethereum blockchain not just as a public bulletin board, but more importantly, as a platform for *consensus computing* that enforces the correct execution of the voting protocol.

In future work, we will investigate the feasibility of running a national-scale election over the Blockchain. Based on the knowledge gained from this paper, we believe that if such a perspective is ever considered possible, its implementation will almost certainly require a dedicated Blockchain. For example, this can be an Ethereum-like blockchain that only stores the e-voting smart contract. This new blockchain can have a larger block size to store more transactions on-chain and may be maintained in a centralised manner similar to RSCoin [9].

8 Acknowledgements

The second and third authors are supported by the European Research Council (ERC) Starting Grant (No. 306994). We would like to thank Nick Johnson for taking the time to answer questions about Ethereum, Solidity and the test-framework Dapple. We thank Maryam Mehrnezhad and Ehsan Toreini for their support in this work during the Economist Case Study Challenge, Malte Möser for his comments on an early draft of the paper, and the anonymous reviewers for their constructive feedback.

References

1. B. Adida. Helios: Web-based open-audit voting. In *USENIX Security Symposium*, volume 17, pages 335–348, 2008.
2. P. Aradhya. Distributed Ledger Visible To All? Ready for Blockchain? *Huffington Post*, Apr. 2016.
3. P. Boucher. What if blockchain technology revolutionised voting? *Scientific Foresight Unit (STOA), European Parliamentary Research Service*, Sept. 2016. http://www.europarl.europa.eu/RegData/etudes/ATAG/2016/581918/EPRS_ATA(2016)581918_EN.pdf.
4. V. Buterin. Long-term gas cost changes for io-heavy operations to mitigate transaction spam attacks. *Ethereum Blog*, Oct. 2016. https://github.com/ethereum/EIPs/issues/150, Accessed on 01/11/2016.
5. V. Buterin. Transaction spam attack: Next Steps. *Ethereum Blog*, Sept. 2016. https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/.
6. J. Clark and A. Essex. CommitCoin: Carbon Dating Commitments with Bitcoin. In *Financial Cryptography and Data Security*, pages 390–398. Springer, 2012.
7. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Annual International Cryptology Conference*, pages 174–187. Springer, 1994.
8. K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, and E. Gün. On scaling decentralized blockchains. In *Proc. 3rd Workshop on Bitcoin and Blockchain Research*, 2016.
9. G. Danezis and S. Meiklejohn. Centrally banked cryptocurrencies. In *23nd Annual Network and Distributed System Security Symposium, NDSS 2016*, 2016.
10. A. Ekblaw, A. Azaria, J. D. Halamka, and A. Lippman. A case study for blockchain in healthcare:medrec prototype for electronic health records and medical research data. 2016. http://dci.mit.edu/assets/papers/eckblaw.pdf, Accessed on 26/10/2016.
11. eth. How do Ethereum mining nodes maintain a time consistent with the network? *Ethereum Wiki*, June 2016. https://github.com/ethereum/wiki/wiki/Light-client-protocol, Accessed on 6/2/2017.
12. Ethereum. Light client protocol. *Ethereum Wiki*, May 2016. https://github.com/ethereum/wiki/wiki/Light-client-protocol.
13. Ethereum. The mix ethereum dapp development tool. *GitHub*, 2016. https://github.com/ethereum/mix, Accessed on 10/10/2016.
14. I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

15. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Crypto'86*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.
16. J. Groth. Efficient maximal privacy in boardroom voting and anonymous broadcast. In *International Conference on Financial Cryptography*, pages 90–104. Springer, 2004.
17. F. Hao, P. Y. Ryan, and P. Zielinski. Anonymous voting by two-round public discussion. *IET Information Security*, 4(2):62–67, 2010.
18. A. Hertig. The First Bitcoin Voting Machine Is On Its Way. *Motherboard Vice*, Nov. 2015. http://motherboard.vice.com/read/the-first-bitcoin-voting-machine-is-on-its-way.
19. S. Higgins. Abu Dhabi Stock Exchange Launches Blockchain Voting . *CoinDesk*, Oct. 2016. http://www.coindesk.com/abu-dhabi-exchange-blockchain-voting/.
20. S. Higgins. IBM Invests $200 Million in Blockchain-Powered IoT. *CoinDesk*, Oct. 2016. http://www.coindesk.com/ibm-blockchain-iot-office/.
21. R. Horrocks. Error while compiling: Stack too deep. *Ethereum Stack Exchange*, June 2015. http://ethereum.stackexchange.com/a/6065.
22. International Association for Cryptologic Research. About the helios system. Oct. 2016. http://www.iacr.org/elections/eVoting/about-helios.html.
23. D. Khader, B. Smyth, P. Y. Ryan, and F. Hao. A fair and robust voting system by broadcast. In *5th International Conference on Electronic Voting*, volume 205, pages 285–299. Gesellschaft für Informatik, 2012.
24. A. Kiayias and M. Yung. Self-tallying elections and perfect ballot secrecy. In *International Workshop on Public Key Cryptography*, pages 141–158. Springer, 2002.
25. R. Kumaresan and I. Bentov. How to use bitcoin to incentivize correct computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 30–41. ACM, 2014.
26. L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 254–269. ACM, 2016.
27. P. McCorry, S. F. Shahandashti, D. Clarke, and F. Hao. Authenticated key exchange over bitcoin. In *Security Standardisation Research*, pages 3–20. Springer, 2015.
28. S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. November 2008. https://bitcoin.org/bitcoin.pdf, Accessed on 2015-01-01.
29. A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and cryptocurrency technologies*. Princeton University Press, 2016.
30. C. Reitwiessner. Smart contract security. June 2016. https://blog.ethereum.org/2016/06/10/smart-contract-security/.
31. A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin. In *Financial Cryptography and Data Security*. Springer, 2016.
32. C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.
33. Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.
34. B. Wire. Now You Can Vote Online with a Selfie. *Business Wire*, Oct. 2016. http://www.businesswire.com/news/home/20161017005354/en/Vote-Online-Selfie.