IEEE ICBC'24 1570981284

Blind Vote: Economical and Secret Blockchain-based Voting

Anonymous Authors

***Abstract*—Electronic voting has been a hot research topic for decades and has recently garnered much attention due to the invention of programmable blockchains that support smart contracts. This is the ideal framework and technology for electronic voting since voting protocols implemented as smart contracts automatically inherit many desired properties from the underlying blockchain, e.g. verifiability, transparency and pseduonymity. However, the public and decentralized nature of the blockchain allows all transactions to be traced by everyone and thus voters' choices would be disclosed publicly. There are many solutions to make blockchain-based voting fully anonymous and untraceable. A recent example is Tornado Vote [1] (ICBC 2023). Such protocols often rely on zero-knowledge proofs, especially zkSNARKS, to achieve secrecy and break the link between a voter's public key and vote. However, verifying these proofs on-chain is expensive and uses a considerable amount of gas (execution fees).**

In this work, we propose a new approach called Blind Vote, which is an untraceable, secure, efficient, secrecy-preserving and fully on-chain electronic voting protocol based on the well-known concept of Chaum's blind signatures. We illustrate that our approach achieves the same security guarantees as previous methods such as Tornado Vote, while consuming significantly less gas. Thus, we provide a cheaper and considerably more gas-efficient alternative for anonymous blockchain-based voting.

***Index Terms*—Electronic Voting, Blind Signatures, Smart Contracts**

I. Introduction

Blockchain. While the blockchain protocol was originally developed to support the Bitcoin cryptocurrency [2], it soon found many further use-cases due to the promising properties it provides, such as decentralization, transparency, and irreversibility. Specifically, it was realized that the same protocol can be used to reach a consensus about the results of any well-defined computation, leading to programmable blockchains such as Ethereum that support arbitrary smart contracts written in a Turing-complete language [3]. This enabled the creation of a wide variety of blockchain-based services and decentralized applications.

Traditional Voting. Voting is a democratic process that requires both confidentiality and accountability. In a typical voting scenario, every participant or third party should be able to verify the result, i.e. the tally, of the process but no participant's choice shall be leaked. In physical voting protocols, voters have to show up in person to cast their ballots and are only informed of the results after a centralized organization performs tallying. Of course, if the voting is for an office, the candidates will each have representatives in the tallying process to create more trust in the system. Nevertheless, this process is opaque and effectively a black box from the point-of-view of the voters and hence undermines voter confidence. See [4] for a more detailed discussion of this point.

Electronic Voting. Designing schemes and protocols for electronic voting has been a hot research topic for several decades. We refer to [5] for an excellent survey. In this work, we are especially interested in blockchain-based voting. This is because smart contracts hosted on the blockchain effectively inherit many of its characteristics, such as verifiability and transparency, which are desirable in a voting protocol. The literature in this domain is vast and there is no way we can do justice to all the previous approaches. Thus, we refer to [6] for a survey of blockchain-based voting methods. We cover some of the most related previous works in Section II. Specifically, the closest previous work is Tornado Vote [1] (ICBC 2023) which provides an anonymous blockchain-based voting protocol based on zero-knowledge proofs.

Our Contribution. In this work, we present a novel blockchain-based voting protocol using a combination of commitment schemes and Chaum's blind signatures as our cryptographic primitives. Our protocol provides the same security guarantees as previous methods, such as Tornado Vote [1]. However, it is important to note that, unlike previous approaches, we purposely avoid zero-knowledge proofs and zkSNARKS. This is an intentional choice to reduce the gas usage (execution costs) of the resulting smart contract. Thus, we present a cheap and gas-efficient anonymous blockchain-based voting protocol without compromising any of the usual security guarantees.

Organization. We provide an overview of the most related prior works in Section II. This is followed by the preliminaries needed for our approach, namely blind signatures and commitment schemes, in Section III. Then, we present our Blind Vote protocol in Section IV. Section V establishes the security properties of Blind Vote. Finally, Section VI compares its gas usage and costs with Tornado Vote [1], illustrating the real-world cost benefits of Blind Vote over the previous state-of-the-art.

II. Related Works

In this section, we consider several of the most related previous works. We refer to [4]–[6] for a more detailed overview of other voting methods.

Desired Properties of an Electronic Voting Protocol. The early work [7], which predates blockchain, identifies the required security properties of a secure electronic voting system as follows (quoted from [7]):

- Completeness: All valid votes are counted correctly.
- Soundness: A dishonest voter cannot disrupt the voting.
- Privacy: All votes must be secret.
- Unreusability: No voter can vote twice.
- Eligibility: No one who is not allowed to vote can vote.
- Fairness: Nothing must affect the voting.
- Verifiability: No one can falsify the result of the voting.

Overview of Previous Works. Many of these properties are attained by default when the voting is implemented as a smart contract. Most importantly, if anyone is eligible to vote, then privacy is achieved by default on blockchain since one cannot associate an account, which is basically a public-private key pair, to a real-life person. However, deanonymization and profiling can still pose threats to privacy [8]. Additionally, in the natural case that we have a predefined set of eligible voters identified by their public keys (accounts), privacy is no longer a given since every transaction on the blockchain is public and traceable. Therefore, *untraceability*, i.e. a disconnect between a voter's public key and their vote, is also needed for a protocol to be secure. We often use the word *secrecy* as a shorthand for untraceability and privacy. There are a wide variety of protocols that aim to achieve secrecy. Examples include the use of homomorphic encryption [9], anonymous off-chain communication channels [7], [10] and, most commonly, standardized tokens and zero-knowledge proofs [1], [11]–[14].

Some approaches sacrifice secrecy or provide a weaker guarantee of privacy. For example, in [7] voters first send their votes to an *administrator* for it to add a signature using blinding techniques. After retrieving the signatures, the voters then forward the votes to a *counter* for it to count the votes and accumulate the results. Although being scalable, this protocol uses an anonymous communication channel as a means to break the link between voters and their votes. However, this practice has two drawbacks: (i) the counter is centralized, and (ii) in the absence of the blockchain protocol, the voters have to perform off-chain communications with the administrator and the counter. These communications can potentially be traced by internet service providers or other intermediaries and used to unmask the voters [15]. Moreover, completeness can be violated if the centralized entities refuse to process valid communications from a voter.

Secrecy via Homomorphic Encryption. The work [9] presents a voting protocol that uses a cryptosystem with an additive homomorphic property to achieve anonymity. The idea is pretty elegant. Here, we provide a simplified outline. In the Paillier cryptosystem, for any two messages m_1 and m_2, we can multiply (aggregate) their encryptions to obtain an encryption of $m_1 + m_2$. This can be adapted to voting in a natural way. An administrator first publishes their Paillier public key on the smart contract. The voters then encrypt their votes using this public key before submitting them to the contract. The contract tallies the votes by multiplying ciphertexts. When the voting is over, the administrator decrypts the final (tallied) ciphertext and hence reveals the final results.

Although the ciphertexts (encrypted votes) are visible all the time on the blockchain, one cannot decrypt them without the private key and hence the votes are secret from the network's point-of-view. However, a lethal drawback of this scheme is that there is always an administrator who should set up the voting and hence possesses the private key. They can always decrypt the ciphertexts off-chain. Thus, there is no secrecy against the administrator. Also, a voter's vote cannot be verified efficiently as it should be encrypted. So, a malicious voter can cast an invalid ballot and affect the overall correctness of the results.

Tornado Vote. Tornado Vote [1] is the most recent and closest related work. It achieves all the desired properties listed above. At its core, Tornado Vote uses a cryptocurrency mixer called Tornado Cash [16] together with zero-knowledge proofs and a relayer infrastructure to achieve secrecy. It uses its own custom ERC-20 for each election. The basic idea is to use zero-knowledge proofs to disconnect voter identities from their votes.

Tornado Vote considers three types of users: administrator, voter and relayer. The role of the administrator is to set up the voting and give eligibility tokens to voters. The role of the relayers, who are often accessed through a secure channel such as Tor, is to send messages to the smart contract on behalf of the voters, ensuring that the source of a message cannot be identified. Since each vote is effectively a token (a piece of currency), voting between k options can be seen as a transfer of money from the voters to one of k predetermined accounts. Mixers, such as Tornado Cash [16], enable such transfers in a way that disconnects the sender and recipient. In a voting setting, this mixing property is exactly the same as the secrecy property, i.e. the sender is the voter and the target account is the chosen vote.

Despite providing all the desired security guarantees, a major drawback of Tornado Vote is its high gas usage. Indeed, the authors make several gas-optimizing choices, such as using different hash functions in various stages, to ameliorate this problem [1]. However, the issue is inherent and already present in Tornado Cash. Its root cause is the necessity of sending zero-knowledge proofs to the smart contract and verifying them on-chain.

In this work, we provide an alternative method which does not require zero-knowledge proofs at all and instead builds upon much more gas-efficient cryptographic primitives such as RSA blind signatures and basic commitment schemes. The result is a huge saving in the overall gas costs of the election.

III. Preliminaries

Our protocol is quite simple and uses only two classical ingredients: blind signatures and commitment schemes.

Blind Signatures. The concept of a blind signature was first introduced by Chaum in [17] to provide both anonymity and privacy to payees in digital cash systems. It allows a payer to obtain a certificate from the bank that blinds it so that the bank can only know the proof of payments but not the identities of the payers. Unsurprisingly, this has already been used for voting, too [18]. However, both the concept of blind signatures and the voting protocols building upon it predate blockchains.

In this work, we use the most classical implementation of blind signatures using RSA [19]. Suppose the bank has an RSA public key (N, e) and its corresponding private key d, and that Alice wants to pay Bob 1 dollar. The protocol goes as follows:

- Alice constructs a banknote, which is a string $b =$ 'This is a banknote with serial XXX...XXX'. The serial number is a random value chosen by Alice. She then computes $h = \text{hash}(b)$ using a pre-defined cryptographic hash function.
- Alice chooses a random number r and keeps it secret. She computes $h' = h \cdot r^e$ and sends it to the bank. Note that, as standard in RSA, all calculations are done modulo N and r^e is the result of encrypting r using the bank's public key e.
- The bank signs h' and sends h'^d to Alice. It also deducts 1 dollar from Alice's balance.
- Knowing r, Alice can easily compute its modular multiplicative inverse r^{-1}. She then obtains the bank's signature on h, i.e. h^d, by a simple calculation:

$$h'^d \cdot r^{-1} = h^d \cdot r^{e \cdot d} \cdot r^{-1} = h^d \cdot r \cdot r^{-1} = h^d.$$

- Alice sends (b, h^d) to Bob.
- Bob immediately sends (b, h^d) to the bank. The bank checks that h^d is a correct signature on the hash $h = \text{hash}(b)$. It also checks that b is well-formed and the serial number in b has not been used before. If the checks pass, it increases Bob's account balance by 1 dollar.

The beauty of the protocol above is that the bank never saw b or h when signing h'. Indeed, knowledge of $h' = h \cdot r^e$ does not give the bank any information about h due to the existence of the random nonce r, which serves as the *blinding factor*. Thus, when presented with (b, h^d) by Bob, the bank is able to verify that b is indeed a valid banknote signed by itself at some point, but it cannot unmask Alice or distinguish her or her banknote from any other banknote of the same denomination.

In the following section, we will develop the idea of using blind signatures to mask voters' identities so as to break the link between the voter and their vote and thus achieve secrecy.

Commitment Schemes. Commitment schemes are a standard cryptographic primitive and often used in blockchain-based protocols. They help mimic simultaneous actions by a group of participants. More precisely, consider n participants who should each send a message to a smart contract. For example, the message can be a bid in an auction. Our goal is to ensure that no participant can know anyone else's message before choosing his own. In the protocol, instead of directly sending a message m, a participant will first hash it with a nonce r to produce $h = \text{hash}(m, r)$. Then, he sends the hash h to the smart contract in the commit phase. The contract records the hash. In the reveal phase, each participant will send (m, r) to the contract, who can in turn compute their hash and ensure the message was not changed.

The simultaneous effect is achieved because hashes in the commit phase leak no information, and hence no one can submit their messages based on any information about the other participants. Moreover, since cryptographic hash functions are collision resistant, one cannot change the message after the commitment phase. In Blind Vote, we will use commitment schemes to mimic simultaneous voting by all eligible voters.

IV. Our Protocol

In this section, we describe our protocol for blockchain-based secret voting using blind signatures. As in Tornado Vote [1], our approach also considers three types of users: an administrator, n voters and a number of relayers.

Our approach consists of the following steps:

Step 0. Deployment. The administrator deploys the Blind Vote contract on the blockchain. During deployment, the following values are set in the contract's constructor (chosen by the administrator):

- The maximum number $n_{\max}$ of allowed voters.
- A registration fee f that has to be paid by every voter;
- A relay reward ρ that will be paid to each relay;
- A deposit δ, which is paid at the time of deployment by the administrator;
- Time limits $t_1 < t_2 < \ldots < t_6$ for the following steps of the protocol. Smart contract functions in each step j of the protocol can only be called after time t_{j-1} and before or on t_j.

The administrator has to ensure that ρ is large enough to cover the gas fees for relays and additionally incentivize them, and that f and δ are large enough for the contract to be able to pay all relays.

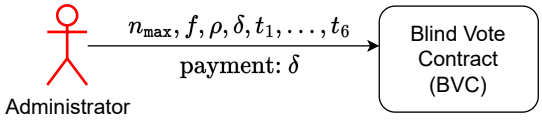

Fig. 1. An illustration of Step 0 of Blind Vote.

Step 1. Voter Registration. This step is open until time t_1. For every eligible voter i who has address a_i on the blockchain, the administrator calls a function `approve`(a_i), adding the voter's address to the voting roll. The contract keeps track of all a_i's. Additionally, each voter should register in the same step, i.e. by time t_1, by calling the `register`() function

in our smart contract and paying a deposit of f. A voter can take part in the remainder of the protocol only if both the registration and approval are done by time t_1. A voter who registers but is not approved by time t_1 can receive a refund after t_1 by calling `step1_refund()`. The contract keeps track of the total number n of valid voters and their addresses and would not allow n to exceed the maximum set in the previous step.

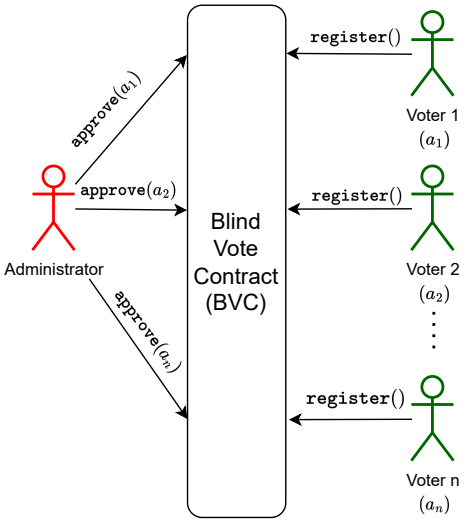

Fig. 2. An illustration of Step 1 of Blind Vote. We use the color red for the administrator and green for voters.

Step 2. Initialization. The administrator generates an RSA public key (N, e) and the corresponding secret key d. He calls the function `initiate`(N, e) of the contract. The contract records N and e, which are now public knowledge. Here, N is the RSA modulus and $x^e \mod N$ is the encryption/signature verification of x. Conversely, $y^d \mod N$ is the decryption/signature on y.

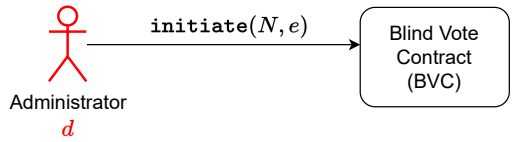

Fig. 3. An illustration of Step 2 of Blind Vote. We show secrets in red and public information in black.

Step 3. Delegation. Each voter i chooses an RSA public key (N_i, e_i) and a corresponding secret key d_i. She keeps all of these values secret for the moment. The voter's goal is to delegate her voting rights to anyone who can sign using d_i, i.e. herself, while making sure that no one can connect d_i to her publicly-known blockchain address a_i. For this, she uses a blind signature scheme as follows:

- Compute $h_i = \text{hash}(N_i, e_i)$.
- Choose a random blinding factor r_i and calculate $h'_i = h_i \cdot r_i^e \mod N$. Recall e is administrator's public key.
- Submit h'_i to the contract by calling `delegate`(h'_i). The contract records the value of h'_i.

The goal is to get the administrator's signature on h_i, i.e. $s_i := h_i^d \mod N$, which serves as a proof that anyone controlling the private key corresponding to (N_i, e_i) can cast a vote.

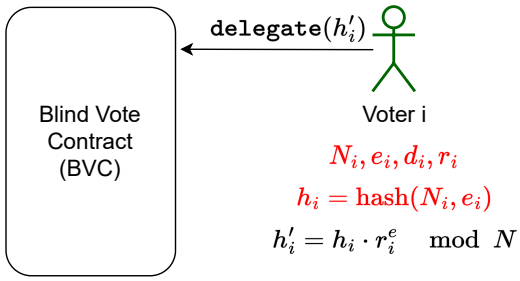

Fig. 4. An illustration of Step 3 of Blind Vote.

Step 4. Blind Signature. For every h'_i provided in the previous step, the administrator computes the signature $s'_i = h_i'^d \mod N$ and announces it to the contract by calling `blind_sign`(a_i, s'_i). The contract checks that s'_i is a valid signature on h'_i and, if so, stores it. The voter i can now unblind the signature on her own machine by computing

$$s_i = s'_i \cdot r_i^{-1} = h_i'^d \cdot r_i^{-1} = h_i^d \cdot r_i^{e \cdot d} \cdot r_i^{-1} = h_i^d \cdot r_i \cdot r_i^{-1} = h_i^d,$$

where all calculations are done modulo N. The latter is the administrator's RSA signature on $h_i = \text{hash}(N_i, e_i)$. Thus, the voter now has the administrator's signature on her own RSA public key without having revealed it to the administrator or anyone else.

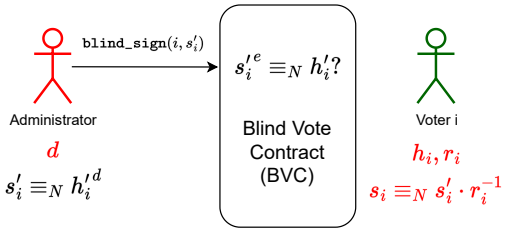

Fig. 5. An illustration of Step 4 of Blind Vote.

At this point, each voter i's ability to cast a vote is delegated to the RSA private key she chose and is no longer connected to her blockchain identity/account address a_i. Specifically, anyone who owns the secret key d_i corresponding to a public key (N_i, e_i) whose hash $h_i = \text{hash}(N_i, e_i)$ is signed by the administrator can cast a vote. In other words, the administrator's signature on h_i is seen as a proof of eligibility for the owner of the corresponding secret key to have one vote in the election.

Relaying. In the next step, voter i will choose her vote v_i. However, she cannot simply send this vote to the contract, since that would (i) leak her identity, and (ii) allow other voters to see her vote before deciding theirs. To overcome (i), we use

the standard technique of *relaying*. A *relay* is a blockchain participant who is willing to submit a function call to the contract on behalf of a voter in exchange for a reward. As is standard, we assume that the voters can send anonymous messages to a public notice board that is seen by relays. We also assume that this does not leak their identity or IP address as they can use services such as Tor to hide this information. A relay can then check if a function call is profitable for them, and if so, is incentivized to make the call on behalf of the voter. Our relaying mechanism matches those of Tornado Vote [1] and Tornado Cash [16]. To solve problem (ii), we apply a standard commitment scheme.

Step 5. Commitment. Each voter i who wants to vote v_i chooses a random nonce x_i and computes $c_i := \text{hash}(v_i, x_i)$. There is a function $\texttt{commit}(N_i, e_i, s_i, c_i, sc_i)$ in the smart contract that can be called by *anyone on the blockchain network*, including relays. This function is used to commit to a particular vote. When this function is called, the contract checks the following:

- The `commit()` function was previously called successfully no more than n times.
- (N_i, e_i) is a valid RSA public key.
- s_i is the administrator's RSA signature on the hash of the public key (N_i, e_i). In other words, $s_i^e = \text{hash}(N_i, e_i) \mod N$.
- c_i is a string that serves as the commitment to a vote.
- sc_i is a valid RSA signature on c_i using the private key corresponding to (N_i, e_i). Formally, $sc_i^{e_i} = c_i \mod N_i$.
- This is the first time this function is called and passed the checks above for the current combination of (N_i, e_i, s_i).

If all these checks pass, the contract records the commitment c_i. It also pays a reward of ρ to the caller of the `commit()` function, who is presumed to be a relay.

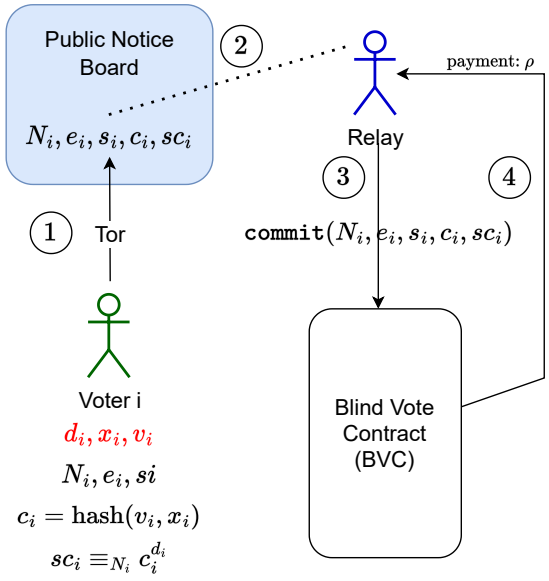

Fig. 6. An illustration of Step 5 of Blind Vote.

Step 6. Revealing. Finally, after all the commitments to the votes are submitted to the contract in the previous section, the voters reveal their votes. This is also done through a relay to preserve their privacy. Specifically, there is a function $\texttt{reveal}(c_i, v_i, x_i)$ which can be called by anyone, including the relays. This function checks the following:

- c_i was a commitment from the previous step and was not revealed before.
- $\text{hash}(v_i, x_i) = c_i$.

If the checks pass, the contract records the vote v_i, updates the tally as necessary, and pays a reward of ρ to the caller of the `reveal()` function who can be the relay[1].

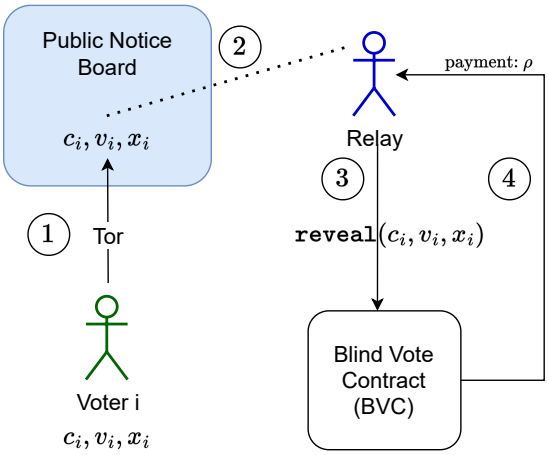

Fig. 7. An illustration of Step 6 of Blind Vote.

If all the steps above are performed correctly, then the votes are submitted to the smart contract using the RSA identities (N_i, e_i) which are blinded and thus disconnected from the voters' actual blockchain identity/account address a_i. So, we have a working blockchain-based protocol for secret voting using only blind signatures and commitment schemes.

Verifications, Incentives and Penalties. To ensure that all parties follow the protocol correctly, we have the following incentive structure:

- After Step 1, any voter who fails to register is excluded from voting.
- After Step 3, any registered voter who fails to successfully call `delegate()` loses the ability to vote, but also her deposit f.
- After Step 4, if the administrator fails to sign one of the h_i' values provided by a voter in the previous step, the voting is canceled and this is seen as cheating. This can be reported by anyone by calling $\texttt{report\_refused\_signature}(i)$. Thus, the administrator's deposit δ is confiscated and paid to the voters. More specifically, each voter i can call a function `step4_refund()` to receive $f + \delta/n$ in her original

[1]It is possible for the voter to call this function herself if she does not care about secrecy. The same applies to `commit()`.

address a_i. This will deter the administrator from cheating and, assuming d is chosen to be large enough, ensures that the voters are compensated for their gas fees and also get their deposits back. Thus, the administrator has to provide exactly n signatures in Step 4 for the protocol to continue.
- In Steps 5 and 6, relays are incentivized to submit the commitment/revealing function calls on behalf of the voters since they will receive a reward of ρ for this.
- After Step 5, if the `commit()` function is successfully called more than n times, this means the administrator cheated and provided extra RSA signatures in addition to the ones in Step 4. In this case, the voting is canceled again, the administrator's deposit is confiscated and paid to the voters. As before, each voter i can withdraw $f + \delta'/n$ into her account a_i by calling `step5_refund()`. Here, δ' is the remaining deposit of the administrator, after subtracting the relay fees.
- A voter who fails to submit her commitment in Step 5 has effectively failed to vote. We assume everyone is naturally incentivized to vote and no one would voluntarily decide not to commit at this step. The same applies to revealing in Step 6.
- If all steps are performed correctly and none of the cases above happen, the administrator can call `admin_refund()` after time t_6 to receive his deposit d back. Similarly, each voter i can call `voter_refund()` to receive a refund of $f - 2 \cdot \rho$, i.e. her initial deposit minus the relaying fees for her messages in Steps 5–6.

Generality of Votes. We note that blind vote can support any system of voting since we are not assuming any particular structure on the votes v_i. Moreover, the tallying can follow any desired formula and the votes do not have to necessarily be a choice out of a fixed set of options. In this sense, our approach is strictly more general than Tornado Vote [1], in which every voter has to choose a vote from a pre-fixed set of possible options. For example, our approach would allow proportional ranked choice voting [20].

Delegation of Voting Rights. In Blind Vote, a voter can easily delegate her voting rights to someone else. In Step 3, the voter i is delegating her voting rights to anyone who has the RSA secret key d_i corresponding to the public key (N_i, e_i). When explaining the algorithm, we presumed that the RSA key pair is generated by the voter herself. However, if she wants to delegate the voting rights to someone else, she can ask them to generate their key pair and only give their public key to her. She will then use this public key in Steps 3 and 4, and provide the unblinded signature s_i to the delegate. Knowing s_i, the delegate takes over Steps 5 and 6 and votes.

Improvements in Gas. There are a number of ways in which we can improve the gas usage of our protocol above, mainly by reducing the amount of storage used by the contract or moving parts of the computations off-chain. We assume that the voters have a secure communication channel with the administrator. We can thus apply the following optimizations:

- *Moving Steps 3 and 4 off-chain.* In Step 3, each voter i sends her h'_i directly to the administrator. This message is authenticated and includes a signature σ_i corresponding to the user's blockchain identity a_i. The administrator then signs h'_i and sends the signature s'_i back to voter i. This whole communication happens off-chain. Only if the administrator fails to provide a valid blind signature s'_i off-chain does the voter call the `delegate()` function on-chain and the administrator will then be required to call `blind_sign`(a_i, s'_i) on-chain, too. If the voter has already received a blind signature on h'_i but then demands another blind signature on a different value h''_i, then the administrator can call a function `report`(i, h'_i, σ_i). This proves that the voter is trying to cheat, allowing the administrator not to provide another signature and also confiscating the voter's deposit.
  This change ensures that, as long as both the voter and administrator are rational and thus prefer not to pay extra gas fees, Steps 3 and 4 can be done off-chain and for free. However, if any party tries to be dishonest, then the normal on-chain protocol will be followed and both will be required to pay gas fees (and potentially also lose their deposit).
- *Premature Commitment.* Suppose the voter has already chosen her vote v_i after Step 2. We can modify the protocol and consider a variant in which the voter does not try to obtain a blind signature on her own public key (N_i, e_i) in Steps 3 and 4, but instead tries to get the administrator's signature directly on her commitment $c_i = \text{hash}(v_i, x_i)$. In this variant, Step 3 will change so that we have $h_i = c_i$. Step 5 will then be simplified with the `commit`(s_i, c_i) function needing access to only s_i and c_i and verifying that s_i is the administrator's signature on c_i, i.e. $s_i^e = c_i \mod N$. While this idea reduces the gas usage, the tradeoff is that it precludes the possibility of delegating the voting rights to a separate person as outlined above.

V. Security Analysis

We now provide brief arguments as to why Blind Vote satisfies all the desired security properties of a secret voting scheme. The most important property, i.e. secrecy, is naturally inherited from blind signatures. Most other properties are inherited directly from the blockchain.

- *Eligibility:* The eligibility to vote is established in Step 1, where the administrator approves all eligible voters. This can also be moved to Step 0, by asking the administrator to provide a hard-coded list of eligible voters. No one other than the eligible voters or the administrator can compute the the blind signatures needed to make a commitment in Step 5. If the administrator cheats and adds extra commitments, there will be more than n valid commitments and the contract penalizes him and cancels the vote. So, there is no incentive for such cheating.
- *Completeness:* As long as the time allocated to each step is sufficiently long to ensure the voters/relayers will be

successful in calling the contract's functions, all valid votes will be committed to in Step 5 and then revealed in Step 6. This ensures completeness.

- *Soundness:* No voter's conduct has any effect on the other voters' ability to vote. A voter who does not follow the protocol correctly can only lose her own voting right / deposit but cannot disrupt the voting.
- *Secrecy:* This important property is inherited from blind signatures. Since each voter's blockchain identity / account address a_i is completely disconnected from the RSA keys she uses to cast her vote by a blinding process in Steps 3 and 4, there is no way to distinguish the source of a particular vote or the vote of a particular voter.
- *Unreusability:* Every voter can obtain only one signature of the administrator on a single hash in Step 4. This signature can then be used only once to commit to a single vote in Step 5. Thus, no voter can vote twice.
- *Fairness:* No one can affect the voting or its results. The administrator is obliged by his deposit to provide the blind signatures correctly in Step 4. As argued, he cannot add extra signatures either. Each voter votes exactly once. A relay cannot affect the results of the voting since they can only get paid their reward ρ if they relay a correct message and everything is verified by the contract. An outside party other than the administrator, voters and relays, has no way of affecting the contract or calling any of its functions.
- *Verifiability:* The blind signatures and commitment schemes are automatically verified by the smart contract and any function call that violates them is automatically rejected. However, since blockchain data is public, anyone with access to the blockchain can separately verify the correctness of the results on their own.

VI. Implementation and Performance Analysis

We have implemented Blind Vote as an Ethereum smart contract written in Solidity. The code is open-source, in the public domain and available at `link removed to preserve anonymity for review.`

As mentioned above, in Blind Vote most of the computations are moved off-chain and hence do not incur gas costs. Moreover, the on-chain computations involve simple and efficient operations such as verifying RSA signatures or computing hashes. We intentionally avoided gas-inefficient operations such as on-chain verification of zero-knowledge proofs. We also store only a tiny amount of information on-chain. To obtain exact gas consumption numbers, we deployed our contract on Remix [21], allowing us to calculate the gas usage of each function call, which is shown in Table I.

Figure 8 compares the gas usage of our approach with 3 previous state-of-the-art blockchain-based voting protocols, namely [1], [11], [22], based on the number n of voters. Our approach significantly outperforms these methods and, assuming that there are 1000 voters, reduces the gas usage by 43.4%, 61.9%, and 83.1% in comparison to Metamask, Tornado Vote and Boardroom voting, respectively. Among these Tornado Vote is the previous state-of-the-art and the only method that provides the same security guarantees as our approach. Moreover, as Figure 8 shows, the improvement gets more pronounced as the number of voters increases.

Step	Function	Min	Max	Paid by
0	constructor	-	6967k	Admin
1	approve	-	71k	Admin
	register	87k	123k	Voter
2	initiate	-	358k	Admin
3	delegate	-	79k	Voter
4	blind_sign	80k	259k	Admin
5	commit	294k	309k	Relay
	commit_premature	119k	210k	Relay
6	reveal	94k	152k	Relay
Refund	admin_refund	-	52k	Admin
	voter_refund	-	83k	Voter
	step1_refund	-	86k	Voter
	step4_refund	-	62k	Voter
	step5_refund	-	76k	Voter
Report	report_refused_signature	-	85k	Voter

TABLE I
THE GAS USAGE OF EACH FUNCTION IN OUR IMPLEMENTATION.

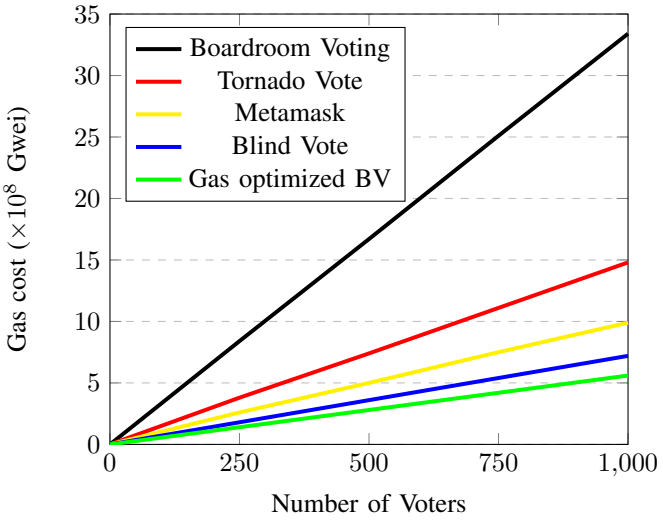

Fig. 8. Gas comparison of different protocols

VII. Conclusion

We presented Blind Vote: a secure and gas-efficient approach for secret voting on the blockchain. Blind Vote uses a combination of RSA blind signatures and commitment schemes to attain all the standard desired security properties of a voting protocol, as well as secrecy, i.e. it is impossible to know which voter cast a particular vote or which vote belongs to a particular voter. We implemented Blind Vote as a free and open-source smart contract and compared its gas usage with previous state-of-the-art blockchain-based secret voting protocols. Blind Vote outperformed these methods significantly in terms of gas usage and obtained improvements of around 40 to 80 percent, hence making blockchain-based secret voting considerably more affordable.

REFERENCES

[1] R. Muth and F. Tschorsch, "Tornado vote: Anonymous blockchain-based voting," in *ICBC*, 2023, pp. 1–9.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[3] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, pp. 1–32, 2014.

[4] T. Moura and A. Gomes, "Blockchain voting and its effects on election transparency and voter confidence," in *International Conference on Digital Government Research*, 2017, pp. 574–575.

[5] M. F. Mursi, G. M. Assassa, A. Abdelhafez, and K. M. A. Samra, "On the development of electronic voting: a survey," *International Journal of Computer Applications*, vol. 61, no. 16, 2013.

[6] F. Þ. Hjálmarsson, G. K. Hreiðarsson, M. Hamdaqa, and G. Hjálmtỳsson, "Blockchain-based e-voting system," in *CLOUD*.

[7] A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections," in *AUSCRYPT*, 1993, pp. 244–251.

[8] F. Béres, I. A. Seres, A. A. Benczúr, and M. Quintyne-Collins, "Blockchain is watching you: Profiling and deanonymizing ethereum users," in *DAPPS*, 2021, pp. 69–78.

[9] S. M. Anggriane, S. M. Nasution, and F. Azmi, "Advanced e-voting system using Paillier homomorphic encryption algorithm," in *ICIC*, 2016, pp. 338–342.

[10] J. C. P. Carcia, A. Benslimane, and S. Boutalbi, "Blockchain-based system for e-voting using blind signature protocol," in *GLOBECOM*, 2021, pp. 01–06.

[11] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in *FC*, 2017, pp. 357–375.

[12] F. Hao, P. Y. A. Ryan, and P. Zielinski, "Anonymous voting by two-round public discussion," *IET Inf. Secur.*, vol. 4, no. 2, pp. 62–67, 2010.

[13] C. Killer, M. Eck, B. Rodrigues, J. von der Assen, R. Staubli, and B. Stiller, "ProvotuMN: Decentralized, mix-net-based, and receipt-free voting system," in *ICBC*, 2022, pp. 1–9.

[14] C. Killer, M. Knecht, C. Müller, B. Rodrigues, E. J. Scheid, M. F. Franco, and B. Stiller, "Æternum: A decentralized voting system with unconditional privacy," in *ICBC*, 2021, pp. 1–9.

[15] A. B. Ayed, "A conceptual secure blockchain-based electronic voting system," *International Journal of Network Security & Its Applications*, vol. 9, no. 3, pp. 01–09, 2017.

[16] A. Pertsev, R. Semenov, and R. Storm, "Tornado cash privacy solution version 1.4," 2019.

[17] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO*, 1983, pp. 199–203.

[18] M. Schmid and A. Grünert, "Blind signatures and blind signature e-voting protocols," *University of Applied Science Biel: Bern, Switzerland*, 2008.

[19] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[20] J. J. Bartholdi III and J. B. Orlin, "Single transferable vote resists strategic voting," *Social Choice and Welfare*, vol. 8, no. 4, pp. 341–354, 1991.

[21] Ethereum Foundation, "Remix – ethereum IDE," 2023. [Online]. Available: https://remix.ethereum.org/

[22] D. Pramulia and B. Anggorojati, "Implementation and evaluation of blockchain based e-voting system with ethereum and metamask," *International Conference on Informatics, Multimedia, Cyber and Information System*, pp. 18–23.