Tornado Vote: Anonymous Blockchain-based Voting

Robert Muth and Florian Tschorsch
Technische Universität Berlin, Germany
{*muth, florian.tschorsch*}*@tu-berlin.de*

***Abstract*—Decentralized apps (DApps) often hold significant cryptocurrency assets. In order to manage these assets and coordinate joint investments, shareholders leverage the underlying smart contract functionality to realize a transparent, verifiable, and secure decision-making process. That is, DApps implement proposal-based voting. Permissionless blockchains, however, lead to a conflict between transparency and anonymity; potentially preventing free decision-making if individual votes and intermediate results become public. In this paper, we therefore present *Tornado Vote*, a voting DApp for anonymous, fair, and practical voting on the Ethereum blockchain. We propose to use a cryptocurrency mixer such as Tornado Cash to reconcile transparency and anonymity. To this end, we adapt Tornado Cash and develop a voting protocol that implements a fair voting process. While Tornado Vote can technically process 10 k votes on Ethereum in approximately two hours, this is not feasible under realistic conditions: Third-party transactions on the Ethereum Mainnet reduce the possible throughput, and transaction fees make it infeasible to use all available block capacities. We therefore present various Gas cost models that yield lower bounds and economic estimations with respect to the required number of blocks and voting costs to assess and adjust Tornado Vote's feasibility trade-off.**

***Index Terms*—anonymous voting, decentralized applications, smart contracts**

I. Introduction

Decentralized applications (DApps) use smart contracts to execute trusted and verifiable logic. The Ethereum [1] blockchain in particular paved the way for novel DApps, such as decentralized autonomous organizations (DAO) [2]. Many DApps hold considerable financial assets and are collectively managed by their stakeholders without delegating decision-making power to centralized bodies [3]. Instead, a blockchain-based voting process is used to coordinate, where the smart contract logic collects votes and ensures that decisions are executed accordingly. That is, anyone can make a public proposal, which can then be accepted or rejected by others. Blockchain-based voting has therefore become an integral part for the governance of DApps.

The inherent transparency of blockchains, however, threatens voters' privacy as it can be considered pseudonymous at best [4]. Anonymous voting is often necessary for democratic decision-making processes, though. While the issue is known for some time and inherent to many blockchains, including Ethereum [5], existing solutions struggle with scalability [6], require a central, trusted off-chain party [7], or cannot directly be used for DApps as they require their own blockchain [8].

In this paper, we propose *Tornado Vote*, a new blockchain-based voting protocol for Ethereum that yields anonymous, fair, and practical on-chain voting. To this end, we build upon the mixer protocol Tornado Cash [9], which enables an anonymous coin transfer service. While Tornado Cash alone can be used for anonymous voting (as we will argue), we adapt the protocol to realize *fair* voting [10] by keeping individual votes secret until all voters have submitted their votes. Additionally, Tornado Vote features (optional) properties such as delegation of voting rights and plural voting. In order to maintain security and anonymity despite our adaption, we use security analysis tools for the smart contract implementations [11], [12] and perform formal code verification, namely VeriSol [13].

In our evaluation, we assess the feasibility of Tornado Vote. To this end, we develop different Gas cost models to quantify theoretical limits and real-world performance bottlenecks. We find that Tornado Vote is technically capable of processing $10\,\mathrm{k}$ votes in 491 blocks ($\approx 2\,\mathrm{h}$) on the Ethereum blockchain under perfect conditions. While this yields a lower bound, it is not realistic in practice. We therefore develop residual capacity models that use historic block sizes to determine the residual capacity of blocks as a more realistic size, which we use to quantify the space for additional transactions, i.e., votes. Since the introduction of EIP-1559 [14], block capacities are variable and transaction fees depend on past transaction loads. This mechanism can lead to unaffordable costs when populating blocks to the maximum. We therefore model the fee calculation of EIP-1559 and develop Gas cost models optimized for a constant fee level. Our evaluation reveals a feasibility trade-off between the number of blocks (or time) for a number of votes and the transaction costs. The cost models can help to find reasonable parameters for this trade-off.

We summarize our main contributions as follows:

- We develop Tornado Vote, a fair, anonymous, and practical on-chain voting protocol
- To this end, we utilize Tornado Cash, an established and effective mixer, and show how it can be used to anonymize votes
- We leverage formal verification and unit tests to secure our implementation
- We develop different capacity models to quantify the feasibility trade-off, including residual capacity models that incorporate historic block capacities and fee adjustments

We introduce blockchain concepts and Tornado Cash in Section II. In Section III, we develop Tornado Vote, present its implementation, and analyze its security. In Section IV, we evaluate different cost models. In Section V, we provide an overview of related work. Section VI concludes the paper.

II. Background

In this section, we first introduce the basic concepts of Ethereum and DApps. Next, we introduce the Tornado Cash protocol, which we later utilize to develop Tornado Vote. Finally, we discuss our ethical considerations regarding the recent U.S. ban on Tornado Cash.

A. Ethereum and DApps

Ethereum [1] and other blockchains with smart contract support enable novel use cases such as decentralized autonomous organizations [2] and decentralized finance lending pools [15]. These use cases can all be summarized under the notion of so-called *DApps*. They are custom programs implemented as smart contracts, which are stored on a blockchain and use it to maintain their own state and coordinate state transitions [1]. DApps therefore profit from blockchain-inherent properties, e.g., decentralization, integrity, and verifiability.

In this paper, we specifically focus on the Ethereum blockchain due to its popularity and its usage of the Ethereum Virtual Machine (EVM), which is also compatible with many other blockchains, though. Since each transaction requires computational resources and storage, transaction fees are required. The computational complexity to execute a transaction and the involved storage consumption is measured in the pseudo-unit *Gas* [1]. In the past, transaction fees used an auction to determine Gas prices, which led to competition for available block capacity and therefore high fees. Recently, Ethereum introduced a new pricing mechanism to calculate Gas prices dynamically, namely EIP-1559 [14]. It uses a congestion control mechanism that regulates a base fee based on available block capacities, which will be burned. Specifically, transaction fees for future blocks increase if the latest block contains "too many" transactions and decrease, respectively, if there is "enough" space up. As a reference for "too many" and "enough", EIP-1559 uses a target value of 15 M Gas (as before), but now also supports temporary breaches up to 30 M Gas to detect congestion.

Similar to other permissionless blockchains, Ethereum stores all transactions transparently and available to enable full verifiability for everyone. This, consequently, poses a severe privacy issue if one does not want to make all account activities public, i.e., income and expenses. There are several techniques for monitoring and clustering transaction activities, even specific for the account-based model used in Ethereum [16]. In order to protect transaction privacy, so-called mixing services emerged that obfuscate the traceability of account activities by unlinking the origin of assets and their current owner.

B. Tornado Cash

Tornado Cash [9] is a smart contract-based, non-custodial mixer for Ethereum's native cryptocurrency *Ether*, standardized tokens (e.g., ERC-20 compatible), and other blockchain assets. For the sake of simplicity, we will refer to all of these assets as *coins* in the following. In a nutshell, multiple accounts deposit coins of the same amount into a shared wallet and withdraw them with a new account in a way that cannot be linked. Anonymity is therefore achieved by hiding in a set of transactions that are indistinguishable, i.e., the so-called anonymity set. Tornado Cash provides such a wallet, i.e., a smart contract, and ensures by using a cryptographic proof that the connection between accounts is not disclosed. At the same time, the proof also ensures that users can only withdraw as many coins as deposited. In order to use Tornado Cash, users do not need to register beforehand and can immediately deposit coins with the first transaction. Users also determine the point in time when to withdraw the coins and therefore can wait for an individually preferred anonymity set size by monitoring the number of deposits.

From a technical perspective, Tornado Cash provides a public smart contract instance, the so-called *vault* and external *relayers*. First, a user deposits a coin from her account to the Tornado Cash vault together with the hash of a personal secret r and a nullifier k. Additionally, the user deposits a fee coverage for future transactions to redeem a relayer in the following steps. Second, the user withdraws the deposited coin to a new account. However, since a completely new account for the withdrawal does not have any balance yet, it cannot cover the transaction fees to request the deposit. The user therefore contacts a Tornado Cash relayer via an anonymized communication channel (e.g., with Tor [17]) and requests the deposit. For that, the user provides a zkSNARK zero-knowledge proof (ZKP) [18] to prove the knowledge and the hash value of k, without revealing r or k in clear-text. The relayer then submits a new blockchain transaction with the ZKP and the hash to the vault to transfer the deposit to the new account. The vault remembers the nullifier and rejects any future request with the same. The vault also rewards the relayer with the pre-paid fee from the first step. Eventually, one cannot reliably trace the depositor back to the withdrawer.

C. Ethical Considerations

The U.S. Department of the Treasury's Office of Foreign Assets Control (OFAC) released a press statement on August 8, 2022, imposing sanctions on Tornado Cash. OFAC's motivation is to prevent money laundering and illegal financing. The sanctions had far-reaching implications, such as banning the use of Tornado Cash, banning trades with specific Ethereum wallets, and the temporary removal of all source code repositories from GitHub. Since Tornado Cash, however, is already deployed on the Ethereum Mainnet, it cannot easily be removed anymore. Operations will therefore continue, particularly since the deployed smart contracts have no assigned owner anymore and do not implement an emergency stop or similar. While our work is technically based on the same technology, the use is intended for anonymous voting—and not for any illegal purposes, which we distance ourselves from. In fact, we consider Tornado Cash to be a neutral technology on whose usage or exploitation we have neither influence nor take a position; rather, we consider its protocol only as building block for blockchain-based voting. However, another suitable mixer or blockchain can be used if necessary, e.g., CoinShuffle [19] or MicroMix [20].

III. Tornado Vote

In this section, we present Tornado Vote, an Ethereum DApp that realizes anonymous, fair, and feasible voting. It is designed specifically with autonomous organizations and the governance of DApps, e.g., DAO [2], in mind. While we leverage blockchain transparency to ensure a secure voting process, we also require on-chain privacy to break the link between a voter and her vote. We therefore utilize the mixing protocol Tornado Cash, which has been shown effective in the past [21]–[23].

In a naive approach, Tornado Cash (or another suitable mixer for that matter) can be used to realize a very simple voting. For instance, voters can use unowned accounts each representing a proposal's option and whose balances represent the votes. Accordingly, voters transfer coins anonymously to these addresses by using Tornado Cash (as explained in the previous section) and eventually compare the balances to come to a decision, e.g., the account/option with the higher balance wins. As illustrated in Figure 1, this approach has the great advantage that other (regular) Tornado Cash users enlarge the anonymity set. This naive approach, however, has some serious, inherent implications: it clearly allows plural voting, where one entity can vote multiple times. While this can be considered as weighted voting, there is no mechanism to easily limit the number of votes per entity. Moreover, the unowned accounts reveal the progress of the voting process and intermediate voting results, which violates the concept of fair voting [10]. Lastly, the approach increases voting costs as it not only requires transaction fees but also transfers coins to a voting account.

In order to make these implications optional and not inherent to the voting process, we developed Tornado Vote. In particular, we extend the voting process with a commit-and-reveal mechanism, which we use to collect votes without revealing the result until the end of the voting, i.e., realizing fair voting. Since we additionally issue our own voting token without inherent monetary value, we can control the number of votes. Yet, plural or weighted voting can still be realized with a voting token by issuing the token according to the respective weights. In the following, we define design requirements, properties, and assumptions of our approach. Next, we present the protocol of Tornado Vote and its different voting phases. Finally, we present the technical infrastructure and security considerations.

A. Design Requirements and Properties

Tornado Vote inherits properties such as correct protocol execution and public verifiability from the underlying blockchain. Nevertheless, there are additional design requirements and properties of Tornado Vote, which we define and elaborate as follows.

Eligibility. Tornado Vote requires a voting token to prove eligibility. More specifically, Tornado Vote requires its own custom ERC-20 token per voting and an initial voting administrator to create and transfer all tokens to eligible voters. Hence, the number of the number of votes is limited by tokens

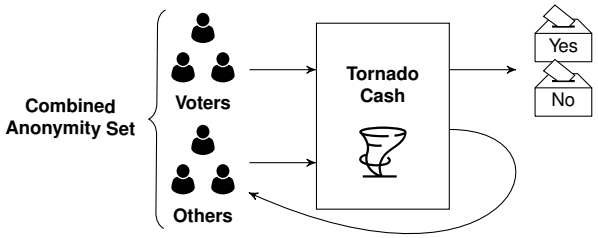

Fig. 1. Utilizing Tornado Cash directly for anonymous voting yields an enlarged anonymity set comprising voters and other users: a voter deposits a coin and anonymously transfers it to one of multiple unowned accounts, whose balances eventually represent the voters' decision.

and managed as standard blockchain tokens. The administrator is responsible for the correct handling at the beginning. After the distribution, the administrator should not own any tokens anymore and should be unable to mint new tokens, which both can be verified on-chain. A voter uses her token to initiate the voting process by depositing it to Tornado Vote. Please note that the voting token does not have any inherent monetary value and therefore does not unnecessarily drive the voting costs as in the naive approach.

Transferability. In some cases, depending on the voting type, it makes sense to allow the delegation of voting rights, e.g., for representative or liquid democracy. Using an ERC-20 compatible token as voting token clearly allows and, to some extent, can even be used to encourage the transfer of voting rights. In fact, preventing transferability is a technical challenge because even if a token cannot be transferred to another account, the whole account could be transferred to someone else by sharing its private key. To this end, if a voting must reliably prevent transferability, the right to vote could be tied to the identity of a voter. For example, by using *anonymous credentials* to prove that a voter is personally eligible to vote regardless of the blockchain account, while keeping all personal information secret [24]. We therefore consider transferability an optional feature that Tornado Vote can enable or restrict.

Plural Voting. While democratic voting typically strives for equal voting weights, shareholders of capital stock, e.g., as in the DAO, gain voting weights according to their stake. We can realize weighted voting by issuing voting tokens to voters according to their stake, for example. This eventually leads to plural voting, an instance of weighted voting. As with transferability, we consider plural voting an optional feature for Tornado Vote.

Fairness. In the context of voting, fairness implies that preliminary counts do not influence voters while voting is in progress [10]. Therefore, the voting system must be built in such a way that votes are not published before the final tallying. That obviously presents a technical challenge for public, permissionless blockchains. Tornado Vote therefore requires a commit-and-reveal mechanism. With this mechanism, voters first commit to their vote and only when all voters have cast their vote, the result will be disclosed as clear-text vote.

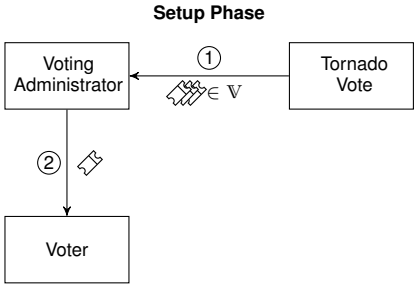

Fig. 2. The administrator mints a limited amount of voting tokens $\mathbb{V}$ and transfers them to eligible voters. Depending on the type of voting, delegation of voting rights and/or plural voting are possible by transferring tokens accordingly.

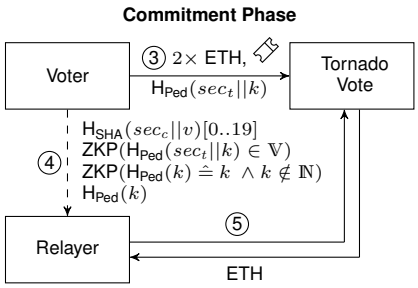

Fig. 3. Voters deposit their voting token, the fees for the relayers, and a hash. Next, voters send ZKPs for proving ownership of the token and a commitment including their vote v to the relayer. The relayer forwards everything and receives a service fee.

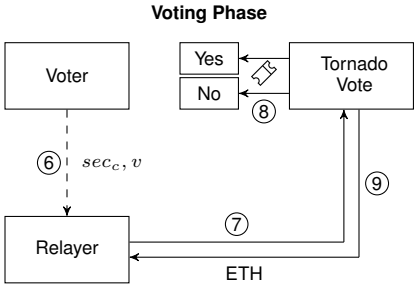

Fig. 4. The voters reveal their commitment and vote to the relayer, who transfers the voting token to the corresponding vote address. The relayer receives a second service fee.

Anonymity. Voter privacy is achieved by the unlinkability of voters' accounts and their votes. To this end, Tornado Vote is based on the Tornado Cash protocol. While the public token balances might reveal the eligible voter accounts, the mixing protocol ensures that voters can anonymize their voting choice. In order to achieve a high level of anonymity, a high number of other participating voters is necessary. As we will describe later in detail, a relayer service is also necessary to cast a vote. The interactions with relayers must be carefully timed to prevent de-anonymization attacks by time correlations, which we assume are the voters' responsibility. We also emphasize that, as with any other public voting process, voter privacy is only guaranteed if the final result is not unanimous. Additionally, network communications between the voters and the relayers must not reveal any metadata that could be correlated. Therefore, voters should use an anonymous channel to communicate with relayers, e.g., with Tor [17].

Trust. Following the principles of a permissionless blockchain, trusted third parties and centralized components must be avoided. While Tornado Cash, and therefore Tornado Vote, require a separate relayer infrastructure, the relayers do not gain any further permissions or trust on the protocol side. Their only task is to forward transactions and cover fees, which will eventually unlink the sender and receiver accounts on chain. Additionally, a relayer cannot manipulate transactions due to their cryptographic signatures and ZKP verification, so that a violation would be noticed and rejected. We assume that sufficient relayers are available to choose from.

Feasibility. Blockchain-based voting systems are limited primarily by the blockchain transaction throughput and smart contract capacities. This limits the number of deposits and thus theoretically affects anonymity. According to an empirical study of blockchain-based voting in [3], we consider $n = 10\,\text{k}$ votes as a reasonable number for use cases such as a DAO. Therefore, Tornado Cash must be built to handle that many votes with respect to external blockchain constraints.

B. Protocol

Tornado Vote distinguishes between three roles: an administrator, voters, and relayers. Each voting is split into three phases: a setup, a commitment, and a voting phase.

Setup phase. First, the smart contracts are deployed to the blockchain by the administrator, which comprises the Tornado Vote anonymity provider, ERC-20 compatible voting token, and the voting smart contract. Second, the administrator publishes the voting proposals and sets voting parameters, e.g., the voting period and decision quotas. Additionally, the administrator mints a limited amount of voting tokens exclusively for this voting. Each token allows the submission of exactly one vote and represents a single vote in the final phase. After this setup, the administrator owns all voting tokens $\mathbb{V}$, as shown in Figure 2, Step 1. The administrator transfers these voting tokens irrevocably to the accounts of eligible voters (Step 2). Once the administrator has transferred all voting tokens, the voting process continues with the commitment phase.

Commitment phase. At the beginning of the commitment phase, voters generate their own individual random token secret sec_t and a random nullifier k locally. As shown in Figure 3 (Step 3), each voter then transfers the voting token to the Tornado Vote vault, similar to a deposit in Tornado Cash. This includes two times the transaction fee (required for the relayer) and the Pederson hash value $H_{\text{Ped}}(sec_t||k)$ [25]. We use the Pedersen hash, as in the Tornado Cash protocol, for efficient ZKP computations. Otherwise, we use SHA-3 where possible to save Gas. After an appropriate waiting time, the voters generate another commitment secret sec_c to commit to their vote v before revealing it. The voters therefore send the first 20 bytes of the hash value $H_{\text{SHA}}(sec_c||v)$, the hash of the nullifier $H_{\text{Ped}}(k)$, and a ZKP for sec_t to a relayer (Step 4). The relayer then forwards everything on-chain (Step 5) to Tornado Vote. Tornado Vote accepts the commitment if the ZKP contains a valid sec_t (without revealing it) and verifies if k was not used before. Therefore, Tornado Vote does not yet experience the voting choice but remembers the corresponding computed hash value and the hash of k in $\mathbb{N}$. Lastly, Tornado Vote rewards the relayer with Ether to compensate for the transaction costs. Once all voters have committed to their vote, we transition from the commitment to the voting phase.

Voting phase. In the last phase, all voters reveal their individual voting choices via a relayer, as shown in Figure 4 (Step 6). To this end, the voters reveal their vote in cleartext and the commitment secret sec_c. This way, Tornado

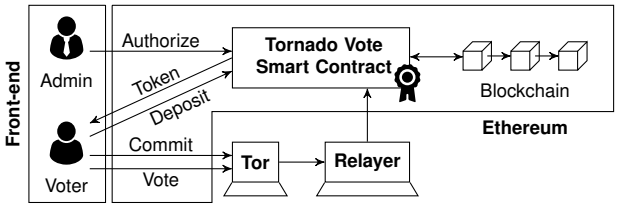

Fig. 5. Tornado Vote's architecture and interactions between its components.

Vote can cross-check that the vote is eligible (Step 7). If so, Tornado Vote transfers one of the ERC-20-tokens to an unowned account, which represents the corresponding voting choice (Step 8). The final balances of these addresses eventually represent the final voting results. After each eligible vote is processed, Tornado Vote again rewards the relayer to compensate for the transaction costs (Step 9).

C. Implementation

Tornado Vote's smart contracts are implemented in Solidity for Ethereum and EVM-compatible blockchains. While Tornado Vote's source code is primarily based on Tornado Cash, it runs independently of it and is deployed entirely on its own. By default, Tornado Vote consists of four components: smart contracts, front-end, relayers, and anonymous communication channel (Tor). As shown in Figure 5, all interactions are managed by a central Tornado Vote instance. For the commitment and vote submission, voters use a Tornado Vote relayer to prevent third parties from linking the two interactions. Furthermore, to protect connection metadata from the relayers, the voter communicates via Tor with the relayers. Internally, Tornado Vote is divided into several smart contracts: the vault (also called *anonymity provider*), Merkle Tree management, ERC-20 token, ZKP verifier, and an additional voting contract. The latter takes care of following the voting phases and counting the votes.

The voters' front-end cannot be used without further ado, as voters require software libraries to generate ZKPs and may download a web-based graphical user interface to interact with Tornado Vote. These software components must first be downloaded, although access to central infrastructures is not a problem here, because file signatures and on-chain proof verification can ensure correctness. Please note that we did not develop a graphical user interface but provide automated test cases which simulate user interactions. The tests cover a sample voting process and edge cases. To this end, they can be executed in a Truffle project environment on the local machine or an existing EVM network. Furthermore, since the Tornado Cash relayer infrastructure is not compatible with Tornado Vote, we provide a proof-of-concept relayer implementation for test purposes. All implementations are available on GitHub.[1]

[1]https://github.com/robmuth/tornado-vote

D. Security

The security of Tornado Vote relies on various components. First of all, the Tornado Vote smart contracts are based on the original Tornado Cash smart contracts. Commissioned audits confirm the security of Tornado Cash's smart contracts, cryptography, and circuit system [21]–[23]. For the security of Tornado Vote, we refer to these audits and therefore focus on our customizations for the voting process. To this end, we use security analysis tools and well-established security libraries. Additionally, we verify our voting process with a formal verification proof framework. In the following, we analyze and improve Tornado Vote's security and explain the technical measures in more detail.

1) Vulnerability Analysis Tools: To ensure that our smart contract customizations do not introduce security vulnerabilities, we use tools for automated code security analysis. Durieux *et al.* [26] analyzed different tools and concluded that *Mythril* [11] and *Slither* [12] offer the best vulnerability detection abilities. We use both tools to detect known vulnerabilities; however, we point out that there may still be yet unknown and undetectable vulnerabilities. Additionally, smart contracts are often so highly complex that the analysis of conditional branches and potentially many possible states cannot be performed in sufficient time. We therefore consider these tools as very useful and to some extent essential for secure smart contract programming. As a result, both tools confirm that our changes do not introduce new security vulnerabilities.

2) Formal Verification: Formal verification is a technique to prove that given rules and conditions are true for a given program, i.e., a smart contract in our case. For example, a condition that proves that a token smart contract cannot issue more tokens than specified at initialization. Thus, if the condition can be fulfilled, it is formally proven that no vulnerability exists to add new tokens arbitrarily. To do so, a formal verification tool must inspect every possible branch in the program logic and verify its states. However, such an excessive state inspection and verification can take a very long time, but if so, it eventually guarantees the condition.

For Tornado Vote, for example, we define conditions that there must be the correct number of votes after each submission. That is, by requiring that a voting counter always increases by one after submitting a vote, we prevent a vote from being ignored, counted twice, or leading to an integer

```
// #Votes == #Tokens
VeriSol.ContractInvariant(VeriSol.SumMapping(_balances) ==
     _totalSupply);

// Optional: Voters can only deposit tokens to Tornado Vote
// in function _beforeTokenTransfer(_from, _to, _id)
if (currentBlock >= commitPhaseBlock && currentBlock <=
     votingPhaseBlock) {
  assert(balanceOf(admin) == 0);
  assert(_to == tornadoVoteAddress);
}
```

Listing 1. Formal verification rule for VeriSol to verify that the number of submitted votes equals the total number of tokens, and an optional check to prevent token transfers to other accounts.

overflow that would reset the counter to zero. To this end, we use formal verification similar to our unit-test cases but universal and without requiring implementation details.

Several formal verification tools and frameworks for smart contracts exist. We use Microsoft Research's VeriSol [13] since it is open-source, compatible with Solidity, and can be executed locally. VeriSol provides a library with additional assertion functions, and verifies all possible outcomes. In Listing 1, the first formal verification rule ensures that the number of possible votes always equals the number of minted voting tokens. To this end, this rule sums up all token balances and compares the sum with the total voting token supply. Additionally, we developed several other assertions for all voting phases directly into Tornado Vote's smart contracts. While most rules can be successfully verified right away, others require constraining parameters or minor smart contract modifications due to code complexity. In summary, we developed and tested 13 conditions with 45 assumptions to verify, from which 11 conditions can be fully verified and 2 require minor smart contract changes. All rules and their verification results are available in our GitHub repository.

IV. Evaluation

Performance and costs are essential factors for voting systems, which in the case of blockchain-based voting lead to a feasibility trade-off. Therefore, we analyze the feasibility of Tornado Vote for ideal best-case scenarios with our *Gas costs model* and compare it with real-world conditions using our *residual capacities model*. We therefore assume that all available block capacities are available for voting transactions and use Ethereum Mainnet data to evaluate realistic capacities. Since our residual capacities model disregards transaction costs, we develop an *economic model* to find a feasible trade-off between maximum performance and minimum costs, which does not cause the transaction costs to skyrocket but still keeps the overall duration reasonable. Importantly, all models are optimistic and should be considered a lower bound for performance and costs.

For clarification, we use the term *Gas costs* to refer to the technical transaction costs in Gas, i.e., the fixed costs due to the transaction's computations and storage usage, and *transaction costs* to refer to the final costs that the transaction sender must pay in Ether or U.S. Dollars (USD), respectively.

A. Gas Costs Model

We evaluate the Gas costs of each Tornado Vote transaction for multiple votes. We, therefore, deploy the smart contracts locally on Ganache and run exemplary voting test cases with up to $n = 250$ votes, as a greater n has no significant effect on the local Gas costs. As shown in Table I, Gas costs for the one-time deployment remain constant for different n, while the transactions for registering eligible voters and submitting votes fluctuate during the following phases (cf. minimums, maximums, means, and the corresponding standard deviation). The Gas costs at the deposit are the most expensive voting transactions, because Tornado Cash internally updates a Merkle tree data structure, and, therefore, causes complex calculations and storage operations. Storage operations are also the main reason for fluctuating Gas costs because, for example, initial write operations to storage variables require more Gas than subsequent operations [1].

TABLE I
Gas costs for deployments and voting transactions.

Transaction	Paid by	Gas Costs			
		Min	Max	Mean	σ
Deployments	Admin	8,192 k	–	–	–
Token transfer	Admin	43 k	58 k	58 k	15 k
Approve	Voter	44 k	44 k	44 k	0
Deposit	Voter	979 k	1,000 k	979 k	21 k
Commit	Voter	337 k	337 k	337 k	0
Vote	Voter	35 k	66 k	50 k	15 k

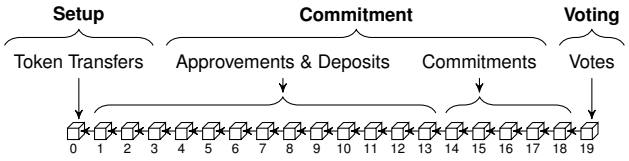

Fig. 6. For $n = 180$ votes, Tornado Vote requires under idealistic conditions (Gas costs model) 20 blocks in Ethereum for all phases.

Using the measured Gas costs, we can estimate the minimum number of blocks for submitting n votes with a static block size limit β (measured in Gas) as follows: $minBlocks(n, \beta) =$

$$\left\lceil \frac{n \cdot 58\text{k}}{\beta} \right\rceil + \left\lceil \frac{n \cdot 1.02\text{M}}{\beta} \right\rceil + \left\lceil \frac{n \cdot 337\text{k}}{\beta} \right\rceil + \left\lceil \frac{n \cdot 50\text{k}}{\beta} \right\rceil \quad (1)$$

Function 1 basically adds the number of blocks in the various phases, or more precisely, the individual steps. To this end, we multiply the number of votes with the Gas costs from Table I and divide them by a static block size limit. We do this for each step in a voting process, i.e., transfer, approve, deposit, commitment, and vote. The approve and deposit steps can be combined and do not require to be finished in separate blocks. Since all other steps must be completed in separate blocks, we yield a total of four terms, which we have to round up to the next integer. For example, a voting with $n = 180$ votes and a typical block size limit of $\beta = 15\,\text{M}$ Gas requires at least 20 blocks. In Figure 6, we also enumerate the number of required blocks for each of the phases and steps.

B. Residual Capacities Model

Tornado Vote's practical feasibility is not only bound to the blockchain's maximum capacities but also to the blockchain's current workload. That is, third-party transactions lead to a fluctuating load, which changes from block to block. Accordingly, the amount of Gas available for the number of votes fluctuates as well. In order to capture this effect, we use a *residual capacities model* for the Ethereum Mainnet as in [3] to estimate the minimum number of blocks for n votes.

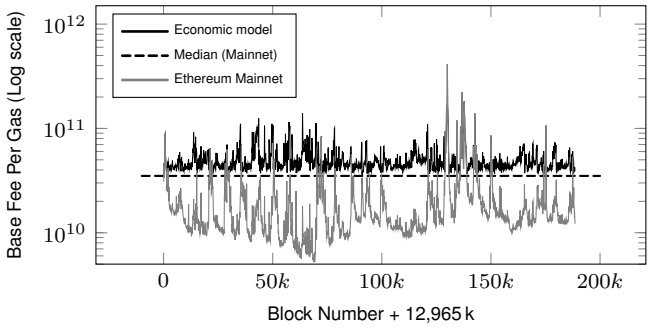

Fig. 7. Base fee per Gas (y-axis) for blocks on the Ethereum Mainnet since EIP-1559 (x-axis) plotting historic Mainnet data and estimations according to the economic model.

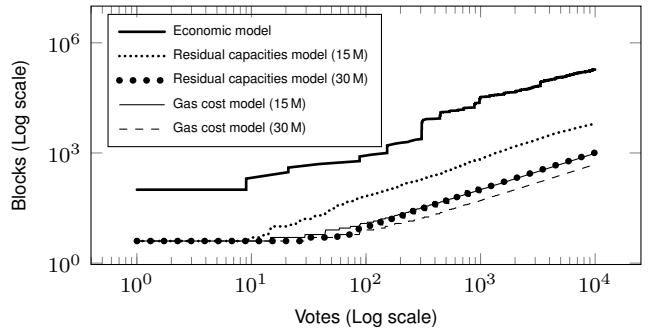

Fig. 8. Minimum number of blocks (y-axis) for an increasing number of votes (x-axis) modelling different block Gas limits (historic data from the Ethereum Mainnet: 2022-11-16 to the past).

Residual capacities are block capacities in Gas that remain unused after a block has been mined. That is, the transactions included in a block did not consume as much Gas as the actual block size limit allowed. We use residual capacities to quantify the number of voting transactions that we could add to blocks without exceeding Ethereum's capacities. Our model therefore sums up all residual capacities from a given point in time towards the past, and calculates how many transactions from Tornado Vote could have been processed despite existing background load.

C. Economic Model

With the London Fork on the Ethereum Mainnet on the 5th of August, 2021, the EIP-1559 specification doubled the block size limit from $\approx 15\,\text{M}$ up to 30 M Gas. The EIP-1559 protocol also introduced the so-called *base fee per Gas* to calculate how much Ether per Gas a transaction sender must pay as a minimum for each block. Basically, this base fee increases if the last block's size exceeded 15 M Gas or decreases if it was less. However, EIP-1559 specifies that the base fee can only change by a maximum of $\pm 12.5\%$, even if the last block size has doubled or halved. This cap provides some confidence to transaction senders that their transactions will not be dropped from the mempool for the next few blocks as long as they provide sufficient Gas. If we, however, would fill up the residual Gas to the max ($\beta = 30\,\text{M}$ Gas) with voting transactions as in our previous models, the base fee would increase by 12.5% each block. As a result, the base fee quickly reaches financially unfeasible amounts. Moreover, even if we utilize the residual Gas only up to $\beta = 15\,\text{M}$, the fees would never go down, but would increase every time third-party transactions exceed the threshold. This also leads to financially infeasible transaction fees, as a matter of course.

We therefore propose an *economic model* that only considers a block's residual capacities for voting transactions if the resulting base fee of the next block does not exceed a given threshold. To this end, we extend our analysis of the Ethereum Mainnet and analyze the past base fees since the introduction of EIP-1559. We show the base fee's development and median value in Figure 7. In the economic model, we consider residual capacities of a block only if its successor block's base fee does not exceed $35\frac{\text{Gwei}}{\text{Gas}}$. Otherwise, the block is omitted. By following our economic model, the base fee will not increase exponentially, but rather settles at the median.

D. Discussion

The presented models allow us to estimate the number of blocks that Tornado Vote requires for n votes under different assumptions. To this end, we analyze Tornado Vote's performance with $n = 1$ to 10 k votes, which we consider a reasonable number of votes for a voting DApp [3]. Since the block Gas limit was doubled to 30 M with EIP-1559, we evaluate our models with $\beta = 15\,\text{M}$ and 30 M Gas separately.

Figure 8 shows the expected trend that an increasing number of votes n (on the x-axis) leads to an increasing number of required blocks (on the y-axis). The Gas costs model constitutes a fundamental lower bound for Tornado Vote since it assumes that all capacities are exclusively available. It quantifies the limit of Tornado Vote in Ethereum, which is limited by Ethereum's transaction throughput. For instance, for $n = 10\,\text{k}$ votes, an optimal coordinated voting with multiple votes per block requires at least 491 blocks with a block size limit of 30 M Gas. Multiplied by an average block generation time $\Delta = 15\,\text{s}$ per block, such a voting would require at least ≈ 2 hours. Accordingly, half the block size limit of 15 M doubles the number of minimum blocks.

Since our Gas costs model ignores the blockchain's current workload, i.e., third-party transactions, we estimate the corresponding number of blocks with our residual capacities model, as well. For that, we measured the residual capacities of the Mainnet from 2022-11-16 until enough residual capacities for 10 k votes accumulated. As shown in Figure 8, utilizing residual capacities therefore requires more blocks, as the blockchain capacities of the Ethereum Mainnet are no longer exclusively available. Interestingly, the residual capacities with 30 M Gas (cf. bold dots) converge to the Gas costs model with 15 M Gas. The reason for this is that EIP-1559 theoretically allows a block size limit of up to 30 M Gas, but the protocol always aims for a block size of 15 M through monetary incentives and penalties; hence, on average, there are always $\approx 15\,\text{M}$ Gas residual capacities that were unused. In the end, the residual capacities model requires at least $\approx 6\,\text{k}$ blocks for $n = 10\,\text{k}$

votes with $\beta = 15\,\text{M}$ and $\approx 1\,\text{k}$ blocks with $\beta = 30\,\text{M}$. In practice, however, these residual capacities cannot be used entirely because the base fee would skyrocket.

Considering the impact of third-party transactions on the base fee per Gas of EIP-1559, our economic model aims to keep the base fee at a financially reasonable threshold. As shown in Figure 7, we therefore analyzed the base fee for past transactions on the Ethereum Mainnet. Using the median base fee as threshold in our economic model, we will approximately maintain this base fee despite our voting transactions. Due to third-party transactions, however, the base fee can temporarily also exceed the median. Following the economic model further increases the number of minimum blocks, which can be seen in Figure 8 (cf. solid bold line). For instance, $n = 10\,\text{k}$ votes require at least $189\,\text{k}$ blocks, which equals approximately 33 days with the same block generation rate Δ as before.

In order to get a sense of the final costs, we estimate the corresponding transaction costs in USD. Therefore, we take the median exchange rate since the introduction of EIP-1559, which was $2{,}780.72\ \frac{\text{USD}}{\text{ETH}}$ at the time of writing (Source: etherscan.io from 2021-08-05 to 2022-11-16). Calculating the minimum Gas for a single vote with the Gas costs model, one vote costs approximately 4.02 USD. Taking the median base fee of $35\frac{\text{Gwei}}{\text{Gas}}$ into account as well, the costs for one vote increases to ≈ 141.06 USD. This rather high price is, of course, only the case if the voting is completed in optimal time, but in any case, we can see that the costs do not increase infinitely, as with the Gas costs or residual capacities model.

Finally, we compare Tornado Vote's costs with the naive Tornado Cash approach (as explained in Section III). To this end, we analyzed all past deposit and withdrawal transactions to Tornado Cash on the Ethereum Mainnet since EIP-1559 for the smallest denomination, i.e., 0.1 ETH. Considering the median of Gas costs for a deposit plus withdrawal transaction, a vote using Tornado Cash directly would cost approximately $1.26\,\text{M}$ Gas or 123.25 USD with the same Gas price and exchange rate as before. If we also consider the additional 0.1 ETH, which has to be sent to vote, the price per vote increases to approximately 401.32 USD. While the coin deposit covers the service fee for the relayer, we assume that for Tornado Vote the relayer service is provided for free, e.g., because the voting administrators have an intrinsic motivation.

Concluding, our model analyses revealed best-case capabilities of Tornado Vote. In practice, filling blocks with voting transactions en masse would have a significant and unpredictable impact on the whole blockchain, though. So, our models cannot predict accurate durations because unpredictable third-party transactions influence or would be influenced by such a large amount of voting transactions. However, we can use our models to assess lower-bound durations and costs.

V. Related Work

In the following, we introduce related blockchain-based voting protocols and point out their key features. While blockchain-based voting is a commonly implemented concept [2], [3], [27]–[29], we focus on *anonymous* voting. McCorry *et al.* [6] implement an anonymous voting protocol based on the *Open Vote Network*. It is implemented for Ethereum and allows up to 40 voters. A self-tallying mechanism computes the final results, but also constitutes a computational bottleneck, which prevents practical implementation in larger-scale voting DApps. While Tornado Vote is bound to a feasibility trade-off as well, we showed that our approach can scale reasonably well to $10\,\text{k}$ votes. In order to improve scalability, Seifelnasr *et al.* [7] introduce an off-chain tallying process for McCorry *et al.*'s voting implementation. Therefore, they rely on at least one trusted instance. Killer *et al.* [8] implement an anonymous blockchain-based voting that is receipt-free. Similarly to Tornado Vote, they also use a mixing protocol. In contrast, however, their proposal requires its own permissioned blockchain and therefore cannot be deployed on Ethereum. By relying on a proof-of-authority consensus, Killer *et al.* achieve significant better scalability and performance than with a permissionless blockchain. Other blockchain-based voting protocols defer to completely private networks [30]–[32]. With Tornado Vote we avoid trusted third parties as all voting processes are implemented on-chain, and relayer infrastructure can and does nothing more than forward transactions, so no further trust is placed in it. Additionally, we maintain compatibility with DApps on the Ethereum blockchain.

For the sake of completeness, we are aware that electronic voting in general has many other weaknesses without further ado, primarily due to other security and privacy issues [33], [34]. We therefore envision Tornado Vote to be used for the governance of DApps, which seems to be able to tolerate some of these trade offs—unlike elections.

VI. Conclusion

In this paper, we designed *Tornado Vote*, a new blockchain-based voting protocol to improve decision-making processes of DApps. It offers anonymous, fair, and practical voting with optional properties such as transferability and plural voting. As a building block, we adapted the well-established mixing protocol Tornado Cash to decouple voters' wallets from their votes. We used smart contract analyzing tools and formal verification to improve the reliability and security of our adaptations. To quantify the feasibility trade-off, we developed different evaluation models that yield Tornado Vote's limits and can help to adjust the trade-off for practical usages.

Acknowledgement

We thank Jorrit Palfner who implemented Tornado Vote as part of his Master's Thesis at TU Berlin.

References

[1] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," accessed 22-12-14. [Online]. Available: https://ethereum.github.io/yellowpaper.

[2] C. Jentzsch, "Decentralized Autonomous Organization to Automate Governance," *White paper*, 2016.

[3] R. Muth and F. Tschorsch, "Empirical analysis of on-chain voting with smart contracts," in *Financial Cryptography Workshops*, ser. LNCS, Springer, 2021.

[4] F. Béres, I. A. Seres, A. A. Benczúr, and M. Quintyne-Collins, "Blockchain is watching you: Profiling and deanonymizing Ethereum users," in *DAPPS*, IEEE, '21.

[5] V. Buterin, *Blockchain voting is overrated among uninformed people but underrated among informed people*, accessed 22-11-15. [Online]. Available: https://vitalik.ca/general/2021/05/25/voting2.html.

[6] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in *Financial Cryptography '17*, LNCS, Springer.

[7] M. Seifelnasr, H. S. Galal, and A. M. Youssef, "Scalable open-vote network on Ethereum," in *Financial Cryptography and Data Security*, Springer, 2020.

[8] C. Killer, M. Eck, B. Rodrigues, J. von der Assen, R. Staubli, and B. Stiller, "Provotumn: Decentralized, mixnet-based, and receipt-free voting system," ICBC, 2022.

[9] R. S. Alexey Pertsev Roman Semenov, *Tornado Cash privacy solution version 1.4*, accessed 22-08-02, 2019. [Online]. Available: https://berkeley-defi.github.io/assets/material/Tornado%20Cash%20Whitepaper.pdf.

[10] S. Delaune, S. Kremer, and M. Ryan, "Verifying privacy-type properties of electronic voting protocols: A taster," in *Towards Trustworthy Elections*, ser. LNCS, Springer, 2010.

[11] B. Mueller, "Smashing Ethereum smart contracts for fun and real profit," 2018. [Online]. Available: https://github.com/muellerberndt/smashing-smart-contracts/blob/master/smashing-smart-contracts-1of1.pdf.

[12] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *WETSEB@ICSE*, IEEE, 2019.

[13] Y. Wang, S. K. Lahiri, S. Chen, *et al.*, "Formal verification of workflow policies for smart contracts in Azure blockchain," in *VSTTE*, ser. LNCS, Springer, 2019.

[14] V. Buterin, E. Conner, R. Dudley, M. Slipper, I. Norden, and A. Bakhta, *Fee market change for ETH 1.0 chain*, accessed 22-11-22. [Online]. Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md.

[15] M. Bartoletti, J. H. Chiang, and A. Lluch-Lafuente, "SoK: Lending pools in decentralized finance," in *Financial Cryptography Workshops '21*, LNCS, Springer.

[16] F. Victor, "Address clustering heuristics for ethereum," in *Financial Cryptography*, ser. LNCS, Springer, 2020.

[17] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Naval Research Lab Washington DC, Tech. Rep., 2004.

[18] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT '16*, ser. LNCS, Springer.

[19] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *ESORICS*, ser. LNCS, Springer, 2014.

[20] B. WhiteHat and K. Gurkan, *MicroMix*, 2019. [Online]. Available: https://hackmd.io/qlKORn5MSOes1WtsEznu_g.

[21] M. Vladimirov and D. Khovratovich, *Tornado cash smart contracts audit*, accessed 22-10-14, 2019. [Online]. Available: http://web.archive.org/web/20220409180200/https://tornado.cash/audits/TornadoCash_contract_audit_ABDK.pdf.

[22] D. Khovratovich and M. Vladimirov, *Tornado privacy solution cryptographic review*, accessed 22-10-14, 2019. [Online]. Available: http://web.archive.org/web/20220409180132/https://tornado.cash/audits/TornadoCash_cryptographic_review_ABDK.pdf.

[23] D. Khovratovich and M. Vladimirov, *Tornado circuit audit*, accessed 22-10-14, 2019. [Online]. Available: http://web.archive.org/web/20220409180142/https://tornado.cash/audits/TornadoCash_circuit_audit_ABDK.pdf.

[24] J. Heiss, R. Muth, F. Pallas, and S. Tai, "Non-disclosing credential on-chaining for blockchain-based decentralized applications," in *ICSOC*, LNCS, Springer, 2022.

[25] Iden3, *Pedersen hash*, accessed 22-12-09. [Online]. Available: https://iden3-docs.readthedocs.io/en/latest/iden3_repos/research/publications/zkproof-standards-workshop-2/pedersen-hash/pedersen.html.

[26] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 ethereum smart contracts," in *ICSE*, ACM, 2020.

[27] K. Garg, P. Saraswat, S. Bisht, S. K. Aggarwal, S. K. Kothuri, and S. Gupta, "A comparitive analysis on e-voting system using blockchain," in *Int. Conference on IoT: Smart Innovation and Usages*, IEEE, 2019.

[28] R. Feichtinger, R. Fritsch, Y. Vonlanthen, and R. Wattenhofer, "The hidden shortcomings of (D)AOs - an empirical study of on-chain governance," *CoRR*, vol. abs/2302.12125, 2023.

[29] X. Xian, Z. Yang, G. Zhang, T. M. de Nelio S., and W. Liu, "A complete anti-collusion mechanism in blockchain," in *BlockSys*, ser. Communications in Computer and Information Science, Springer, 2020.

[30] B. Yu, J. Liu, A. Sakzad, *et al.*, "Platform-independent secure blockchain-based voting system," in *ISC*, LNCS, Springer, 2018.

[31] F. S. Hardwick, A. Gioulis, R. N. Akram, and K. Markantonakis, "E-voting with blockchain: An e-voting protocol with decentralisation and voter privacy," in *iThings/GreenCom/CPSCom/SmartData*, IEEE, 2018.

[32] F. P. Hjalmarsson, G. K. Hreioarsson, M. Hamdaqa, and G. Hjálmtýsson, "Blockchain-based e-voting system," in *IEEE CLOUD*, IEEE Computer Society, 2018.

[33] S. Heiberg, I. Kubjas, J. Siim, and J. Willemson, "On trade-offs of applying block chains for electronic voting bulletin boards," *IACR Cryptol. ePrint Arch.*, 2018.

[34] W. Bokslag and M. de Vries, "Evaluating e-voting: Theory and practice," *CoRR*, vol. abs/1602.02509, 2016.