

Privacy-Preserving Inclusion Lists

Zhengwei Tong   

Duke University, Durham, NC, USA

Saba Eskandarian   

University of North Carolina at Chapel Hill, NC, USA

Kartik Nayak   

Duke University, Durham, NC, USA

Abstract

Blockchains aim to provide open access and censorship resistance, but centralization of block production in blockchains like Ethereum undermines these goals. Inclusion List (IL) protocols mitigate this by requiring block proposers to include transactions selected by an IL committee to enforce the inclusion of transactions that appear to have been censored. However, protecting the confidentiality of individual committee members' contributions is essential to prevent retaliation and ensure robust censorship resistance.

We propose a lightweight, privacy-preserving inclusion list protocol that allows committees to collectively construct transaction lists while hiding individual contributions and ensuring plausible deniability. Our approach builds on multiparty computation (MPC) techniques to achieve strong privacy without relying on heavyweight cryptography or anonymous broadcast channels.

We implement two variants of our protocol design: an optimistic version providing malicious security with abort (latency $\sim 4.0s$) for speed, and a robust variant (latency $\sim 124.7s$) for guaranteed output delivery in the presence of a Byzantine threshold of $t < n/3$ malicious parties.

2012 ACM Subject Classification Security and privacy $\rightarrow$ Privacy-preserving protocols

Keywords and phrases Blockchain, Inclusion List, Privacy, Multiparty Computation, Censorship Resistance

Digital Object Identifier 10.4230/LIPIcs.AFT.2026.2

Related Version *Full Version*: <https://eprint.iacr.org/2026/1582> [32]

Supplementary Material *Software (Source Code)*: <https://github.com/timetzw/Privacy-Preserving-Inclusion-List>, archived at `swb:1:dir:55cf3bee05b224f078f2c18b98fe29e5d40b6f55`

Funding This material is based upon work supported by the National Science Foundation under Award Nos. 2234408 and 2237814. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

Blockchains provide the ideal functionality of a general-purpose computer that is available and accessible to anyone, has no owner or operator, and supports property rights for user-owned data. However, implementations differ from such an idealized world, and there exists forces of centralization due to which these goals are not achieved. For instance, many blockchain protocols use block proposers or leaders that are responsible for selecting transactions added to the blockchain; thus, at any time, they temporarily control the availability and accessibility of the blockchain. In fact, in blockchains such as Ethereum, block production is outsourced to entities called builders, and the top two builders are responsible for producing more than 90% of blocks [39, 38, 29]. Ironically, this is similar to the centralized credit card networks where 95% of the purchase volume is controlled by the top three networks (Visa, Mastercard, and American Express) in the United States [27].



© Zhengwei Tong, Saba Eskandarian, and Kartik Nayak;
licensed under Creative Commons License CC-BY 4.0

8th Conference on Advances in Financial Technologies (AFT 2026).

Editors: Aggelos Kiayias and Maria Kyropoulou; Article No. 2; pp. 2:1–2:23

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A key concern related to the accessibility of the blockchain is censorship: a valid transaction proposed by any party must be included in the chain without undue delay, and censorship is the act of refusing to include valid transactions on the chain. There are two key reasons for censorship: self-censorship due to regulation and financial incentives. In recent years, we have observed self-censorship in the form of OFAC sanctions [33]. Moreover, applications related to decentralized finance can create financial incentives for censoring [36, 37, 17]. However, for a system intended to be accessible to anyone and everyone, we need to ensure that transactions cannot be censored by potential centralizing entities. To this end, parties that uphold the ideal functionality and provide censorship resistance should not be at risk of exposure or harm for doing so.

In blockchains such as Ethereum, recent proposals for addressing censorship concerns have focused on building a separate sidecar protocol called an *inclusion list protocol* [31, 35]. These protocols produce an agreed-upon list of transactions that the block proposer is then required to include in its proposed block, or else the block is not considered valid [30].

A key approach to prevent censorship in the inclusion list protocol itself is to use a committee of *multiple proposers* (also called IL proposers) and to output transactions that are proposed by a threshold of the proposers. Having multiple parties involved in producing the inclusion list ensures greater decentralization, so it is harder for transactions to be left out if a threshold of the parties are honest. Consequently, the job of the honest parties in the committee is to propose transactions in the mempool that are being censored.

1.1 Privacy and inclusion lists

Unfortunately, the act of combating transaction censorship by serving on an inclusion committee comes with risks. A member of an inclusion list committee may run afoul of censoring entities by forcing inclusion of transactions that would otherwise be censored. Thus, we aim to protect the confidentiality of which committee members proposed which transactions for inclusion in the final inclusion list. This is akin to the principle of protecting confidentiality in voting systems.

Adding privacy to inclusion list protocols safeguards committee members against punishment for supporting censorship resistance, and parties can uphold the ideal properties of the protocol without the risk of exposure or retaliation [1]. The goal of this work is to design such a multi-proposer privacy-preserving inclusion list protocol that incorporates transactions proposed by a threshold of parties while also protecting the privacy of parties proposing these transactions.

1.2 Key challenges and desiderata

There are several key challenges to designing such a privacy-preserving inclusion list. A core challenge involves achieving the desired privacy property that de-links a transaction input by a party from the output of the committee, even if some parties deviate from the protocol and behave arbitrarily. Thus, a proposer obtains the necessary anonymity guarantees and it can plausibly deny having included any transaction. Crucially, these guarantees must also hold at the network level, i.e., we want to achieve these guarantees even against an adversary that is able to observe all network traffic.

The main performance goal for an inclusion list protocol is to ensure potentially censored transactions can be included on a blockchain quickly. The lower the latency of the inclusion list protocol, the sooner the censored transactions can be included in the main chain. Thus, if it takes time equivalent to k blockchain “slots” to produce one inclusion list, then our goal is to reduce this k ; ideally $k = 1$ and we are able to include a new inclusion list in every time slot.

Prior private inclusion list proposals [25, 1, 2] have made use of linkable ring signatures [5, 21] used on top of an anonymous communication network [15, 9, 20, 10, 11]. Unfortunately, these proposals have not been fully specified, analyzed, or implemented, and their security relies on a sufficiently performant anonymous communication system, a problem where significant performance and anonymity tradeoffs are inherent [13, 14]. We compare the setting and security properties of our proposed scheme with prior proposals in Section 7.

1.3 Our contribution

We present a protocol that allows an inclusion list committee to produce an inclusion list collectively without revealing any individual member’s contributions and while giving each member plausible deniability for introducing each transaction to the final inclusion list. Our protocol is agnostic with regard to committee selection processes, assuming only that committee members can communicate with each other, and it does not require a network-level anonymity mechanism.

Our protocol employs ideas from the literature on multiparty computation (MPC) [19, 3, 12, 23, 22] and private computation of heavy hitters [6] to allow committee members to develop a list of transactions that many members want to place on the inclusion list. Apart from the MPC protocol, which computes simple functions like private sums or products, the protocol only uses lightweight, standard cryptographic tools, e.g., hashing and signatures.

Conceptually, the protocol operates over the prefix tree of all possible transaction hashes, but it explores this tree lazily: at any point, the parties maintain only the active prefixes that have not yet been pruned. At each layer of the tree, they compute a private sum for each node that lies on a path to a transaction they want included in the final list. Nodes with sums that are too low are pruned from the search, allowing the process to run efficiently, and any leaves that have not been pruned by the end of the protocol constitute the transaction hashes that correspond to the chosen inclusion list. Computing all sums privately under an MPC ensures that it is never known which committee members “voted” for which transaction, even if their protocol messages are visible to attackers at both the application and network layers. Our full protocol augments this idea with additional techniques to ensure stronger plausible deniability, protect against malicious adversaries, and improve performance.

1.4 Implementation and evaluation

We implement two versions of our protocol. Both provide all necessary privacy properties, but they differ in output delivery guarantees. The first is an optimistic version that uses the Mascot protocol in MP-SPDZ [23, 22] and completes execution only when all of the committee members cooperate. When evaluated under parameters chosen consistent with EIP-7805 [30] and a 16-member IL committee, the protocol runs within 4.0 ± 0.1 s. The second version is robust under a Byzantine threshold of $t < n/3$ committee members, but runs in 124.7s.

The optimistic protocol completes well within Ethereum’s 12-second block timing constraints, while the slower robust variant requires several blocks to produce an inclusion list. This suggests the possibility of running both protocols in parallel, allowing for quick inclusion list results in the common case, and eventual results in the slower case. We present a detailed analysis in Section 6.

2 Background and Preliminaries

2.1 Notation

We use $x := f(y)$ to denote assignment to x of the value $f(y)$, and $x \xleftarrow{R} S$ to denote assignment to x of a value chosen uniformly at random from a set S . Depending on context, we use either subscript notation or array notation to access an item on a list, i.e., vals_i or $\text{vals}[i]$. A function $\text{negl}(x)$ is *negligible* if, for all $c > 0$, there exists an x_0 such that, for all $x > x_0$, $\text{negl}(x) < \frac{1}{x^c}$.

2.2 Background: Inclusion Lists (IL)

Inclusion lists were introduced to address the censorship risks that emerged after Ethereum’s transition to Proposer–Builder Separation (PBS). With PBS, proposers outsourced block construction to specialized builders through an auction. This market structure created a centralization pressure: when the builder market becomes dominated by a small number of powerful actors [39], those builders effectively have the power to selectively censor transactions.

After a series of efforts to understand and address the problem of transaction censorship [7, 24, 8, 28], Fork-Choice enforced Inclusion Lists (FOCIL)[31] emerged as a solution that addressed concerns about incentives in prior proposals. FOCIL improves on earlier approaches in two ways: first, by replacing a single inclusion-list proposer with a committee to reduce susceptibility to bribery; and second, by enabling inclusion-list enforcement within the current slot, eliminating the one-slot delay required in prior designs.

FOCIL inclusion list committees gossip their views of the transaction mempool to come to a consensus on a list of transactions may have been censored, and the resulting list of transactions must be included in the next block. This paper augments this process with stronger privacy protections for committee members. While addressing the pros and cons of this kind of inclusion list design in general are out of scope for this work, we briefly discuss inherent limitations of inclusion list designs in Section 7.

2.3 Multiparty computation (MPC)

Multiparty computation (MPC) protocols allow parties with private data to jointly compute a function without revealing their individual inputs to each other. Conceptually, this is like a trusted third party collecting secrets and returning the result, but in practice, the parties run a protocol that simulates this role. Inputs are secret-shared – split into pieces $[x]_1, \dots, [x]_n$ such that $\sum_i [x]_i = x$ – and operations on these shares allow secure addition, multiplication, and thus any arithmetic circuit. At the end, parties combine their shares to reveal the output. We express our protocol by describing the function $\mathcal{F}_{\text{MPC}}$ that the trusted party would compute based on the parties’ inputs; in our implementation, this function is realized by replacing it with an actual MPC protocol.

2.4 MPC threat models

MPC protocols exist for a variety of threat models. We consider a static, malicious adversary controlling up to t parties. Our security analysis distinguishes between two notions of security against malicious behavior.

- **Malicious security with abort:** The adversary can corrupt the internal state and messages of corrupted parties. We guarantee that the protocol either outputs the correct result of the functionality $\mathcal{F}_{\text{MPC}}$ or aborts. Malicious security with abort can be achieved with up to $n - 1$ of n parties being malicious.

- **Guaranteed output delivery:** This variant guarantees that, assuming at most $t < n/3$ Byzantine parties, eventual message delivery among honest parties, and the standard assumptions of the underlying MPC protocol, all honest parties eventually output the same valid result. In particular, corrupted parties may deviate arbitrarily from the protocol, but they cannot prevent honest parties from obtaining an output.

3 Problem Description

Our core problem involves a set of n parties $P_1, \dots, P_n$. We assume that up to t of the n parties are Byzantine, and thus can behave arbitrarily; the remaining parties are honest. Each of the n parties can communicate with each other using an authenticated channel.

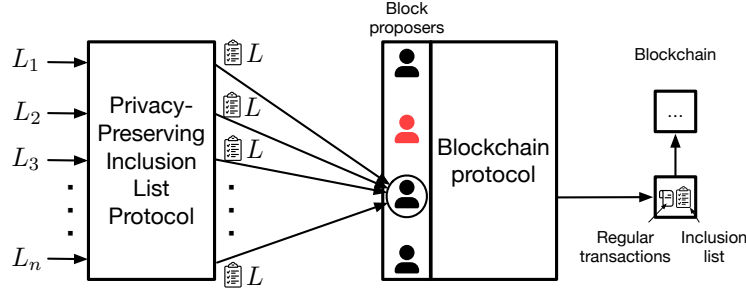
There is a global set of values representing transaction hashes, where each value is a κ -bit value chosen uniformly at random. Each party P_i has an input list L_i containing a subset of these values. Our goal is for the parties to privately compute an intersection of all parties' input lists, and output a list L consisting of values that appear in a threshold number of inputs.

3.1 Security Goals

Intuitively, the privacy goal is to ensure that an adversary cannot determine whether a specific party input a specific value based on the protocol transcript and the final outputs. More precisely, we want the following properties to hold for parameters y and t :

- **Correctness.** We use two correctness notions, depending on the output-delivery guarantee of the MPC backend.
 - **Optimistic correctness.** If a value v is included in the inputs of at least y honest parties and the execution completes without abort, then v must appear in the output except with negligible probability in λ .
 - **Robust Correctness.** If a value v is included in the inputs of at least y honest parties, then despite arbitrary behavior by the t Byzantine parties, v must appear in the output except with negligible probability in λ .
- **Soundness.** No value v will be included in the output L unless it is included in L_i for at least one honest party.
- **Anonymity.** Parties to the protocol learn at most the count of how many parties have included a value v in their list L_i , with no information leaked about which parties contributed to the count.
- **Plausible deniability.** For any adversary corrupting up to t parties, any honest party P_i and any value v , the adversary cannot rule out an execution possibility consistent with its view in which P_i did not contribute v to the protocol.

Anonymity and plausible deniability are related but distinct. Anonymity hides which parties contributed to a revealed aggregate count. However, if the protocol revealed that all committee members contributed to a value v , anonymity alone would not let any party deny contributing v . Deniability addresses this edge case by requiring that, for any given committee member, the transcript and output do not rule out the possibility that the member did not contribute v . We include the plausible deniability definition to augment anonymity, and our proposed protocol satisfies both requirements.



■ **Figure 1** Privacy-Preserving Inclusion List protocol, the protocol should produce a singular version of the inclusion list to the next proposer in the ideal case.

3.2 Design Considerations

We now describe the larger context in which solving the above problem is relevant (cf. Figure 1). As described in the Introduction, in blockchains such as Ethereum, there are designated proposers for each *slot* of the protocol. Each slot can hold a block, and as of today, more than 90% of these blocks are built by two external entities called builders [39, 38, 29]. Such centralization makes it easy for transactions to be censored, either due to economic reasons or due to coercion [33, 36, 37, 17]. The notion of an inclusion list is meant to address this problem by relying on a decentralized set of blockchain validators to agree on a set of censored transactions, and the protocol requires the block proposer to include the agreed-upon set in the next slot.

We rely on the existence of a committee of validators whose members hold responsibility for producing the inclusion list for each slot. The membership of the committee is public, and there must be an agreement around who is on the committee. While the mechanism for constructing the committee is outside the scope of this work, our proposed solution is compatible with any committee that satisfies these requirements. However, for decentralization purposes, it will be beneficial if the committee is chosen at random afresh for each slot.

The committee members form the n parties we referred to in the problem statement. We assume each of these parties must have access to the mempool over time in order to determine which transactions should go on an inclusion list. This is the reason why we assume each of the parties' inputs comes from a global set of values. We note that no individual party may know the exact global set. While there will naturally be a large degree of overlap between committee members' views of the mempool, we do not require them to have identical views.

3.2.1 Protocol inputs and outputs

The committee members provide *transaction hashes* as input, and at the end of the protocol, each member outputs the agreed-upon inclusion list of transaction hashes. Committee members sign the list of hashes and share them with proposers on the network. A list signed by more than t committee members (and thus, at least one honest committee member) is considered a valid inclusion list for the slot. In order to produce a valid block, proposers must show that for each transaction hash h on the inclusion list, they have included a transaction tx such that $H(\text{tx}) = h$ for the hash function H used by our scheme. Note that signing the outputs of the protocol is not connected to each committee member's individual inputs, so signatures on the output of the protocol do not reveal which, if any, of the included transactions were added by any individual committee member.

The hash function H can be any collision-resistant hash function of appropriate output length. Since the performance of our core protocol will depend on the output size of the hash function used, we can optimize performance by setting the output length to be shorter than is typically used for CRHFs, e.g., 64 bits, without affecting the privacy properties of the scheme. This introduces the possibility of occasional collisions where two different transactions hash to an entry on the inclusion list. We consider a block with either transaction to be valid, with the understanding that if the “wrong” transaction is included, the protocol execution on the next slot will still include that hash on the inclusion list, and the “correct” transaction can be included the next time around. In Section 6, we explore parameter tradeoffs like this one and we introduce optimizations in Section 5.4 that replace a single hash with multiple shorter hashes in order to improve parallelism in protocol executions.

3.2.2 Security properties in the larger context

We make a few observations relating to the properties described in the problem statement and the larger context.

- First, our core protocol outputs transaction hashes rather than full transaction payloads. This keeps the MPC lightweight, but it creates an additional availability requirement: a block proposer who receives a hash h must still obtain some transaction tx such that $H(tx) = h$. Soundness ensures only that an included hash was supported by at least one honest committee member, and therefore that a matching transaction exists in at least one honest party’s view of the mempool. Looking ahead, our protocol will test that any transaction output has been included by at least some threshold $\tau \geq t + 1$ number of parties. We note that, if payload availability was ensured through other means or it was not needed, we could avoid the constraint on τ . See discussions in the full version [32].
- Second, our correctness property guarantees that a transaction input by at least y honest parties must appear in the output. Given censorship is a motivation, our goal will be to minimize the value of y .
- Finally, our requirements of anonymity and plausible deniability ensures that parties can input censored transactions while remaining unlinkable to the transactions that they input individually (anonymity) and each being able to deny that they input any particular transaction (plausible deniability).

3.3 A Two-Track Approach

Our core technical contribution is a multiparty protocol for voting on and producing a list of transaction hashes that constitute the contents of an inclusion list. The protocol, described in Section 4, consists of a recursive voting/narrowing procedure over the space of all possible hashes that identifies which hashes are included in the individual inclusion lists of τ or more committee members, without revealing which members have which transactions on their lists.

A large part of our core protocol involves running a series of small MPC subprotocols. The underlying MPC protocols used must provide malicious security and robustness due to the possibility of malicious parties on the committee tampering with the protocol or launching a denial of service attack by going offline. Unfortunately, known robust MPC protocols are more expensive than their counterparts that only provide malicious security with abort. The latter provides the guarantee that if an honest party gets an output, then it can check that it has indeed received the correct output, but it leaves room for malicious parties to interrupt the protocol by simply aborting [23, 12].

We observe that since malicious security with abort suffices for our soundness and privacy requirements, we can adopt a two-track approach. We run two versions of the protocol in parallel. The first version is a faster MPC protocol that only provides security with abort, i.e., satisfies optimistic correctness, and outputs a value within a deterministic time bound under a synchronous network. The other version provides correctness even in the face of arbitrary behavior by the t malicious parties. If the first version runs successfully, the output of that version will be considered for the slot; otherwise, we consider the output of the robust version, which will become available for a future slot.

4 Protocol Description

This section describes the core technical contribution of this paper: a privacy-preserving inclusion list protocol that satisfies the design goals and security requirements set forth in Section 3.

Prior approaches to anonymous inclusion lists rely on ring signatures and assume an underlying anonymous broadcast protocol, separating out application and network level privacy concerns [25, 1, 2]. We observe that by collapsing these two levels of the protocol stack, we can construct solutions that simultaneously use more lightweight cryptographic primitives and provide tailor-made privacy guarantees, reducing the demands placed on the underlying communication protocol. Our protocol will only require standard encrypted/authenticated channels between parties.

We adapt ideas from multiparty computation [19, 3, 12, 23, 22] and anonymous communication [9] to produce lightweight privacy-preserving inclusion lists. Our high-level approach is inspired by private heavy-hitters protocols, which recover popular strings from many private client inputs without revealing any individual client’s string [6]. In our setting, the “popular strings” are transaction hashes supported by many committee members. Since the protocol itself hides who contributed which transaction, we do not need a separate anonymity layer to hide who sends which message within the protocol.

We also wish to satisfy a deniability property: any committee member should be able to plausibly argue that they did not add a given transaction to the inclusion list. Our scheme leaks only the (approximate) number of members who included each transaction on the final inclusion list. This is because the protocol allows the committee, subject to Byzantine inputs, to enforce arbitrary thresholds on how many members need to include a transaction for it to be on the list.

We discuss our solution below, first assuming a fully semihonest setting where all parties follow the protocol but corrupt parties wish to learn information based on the protocol transcript. Next, we will sketch how to lift the security of our protocol to tolerate malicious behavior.

4.1 A Semihonest Secure Approach

We think of inclusion list construction not as a problem where a number of parties append transactions to a list, but rather as a problem of selecting which transactions from the list of all possible transactions should be included. Intuitively, this is the difference between a sushi restaurant waiter asking for each person’s order versus a table filling out a sheet with their menu selections. In the former scenario, each order is directly tied to the customer who asked for it, whereas in the latter, the group as a whole is connected to all parts of the order. Loosely speaking, our protocol will allow a committee to produce the completed order sheet without leaking any one member’s contribution.

```

1:  $L_i$ : list of input transaction hashes for party  $P_i$ 
2:  $AP_l$ : Set of active prefixes at level  $l$ 
3:  $b$ : branching factor, a power of 2, representing the number of children of one node to explore
4:  $\kappa$ : length of input transaction hashes
5:  $n$ : number of parties
6:  $\tau$ : threshold
7:  $x$ : maximum permissible number of zero-input parties
8:  $\text{coin}_i \xleftarrow{\mathbb{R}} \{0, 1\}^\lambda$ 
9:  $\text{out}_{\text{coin}} := \mathcal{F}_{\text{sum}}(\text{coin}_1, \dots, \text{coin}_n)$ 
10: Let  $j$  denote the smallest index in  $\text{out}_{\text{coin}}$  s.t.  $n - x \leq \text{out}_{\text{coin}}[j] \leq n - (t + 1)$ 
11: if  $\text{coin}_i[j] = 0$  then ▷ zero-input party
12:    $L_i = \{\}$  ▷ Sets input to be empty
13:  $AP_0 := \{0, \dots, b - 1\}$ 
14:  $d = \log_b(2^\kappa) - 1$ 
15: for level  $l := 0, \dots, d - 1$  do
16:    $V_{i,l} := \{\}$  ▷  $P_i$ 's vote map for level  $l$ 
17:   for prefix  $p \in AP_l$  do
18:     if  $\exists tx \in L_i$  with prefix  $p$  then  $V_{i,l} := V_{i,l} \cup (p, 1)$ 
19:     else  $V_{i,l} := V_{i,l} \cup (p, 0)$ 
20:    $V_{G,l} := \mathcal{F}_{\text{MPC}}(V_{0,l}, \dots, V_{n-1,l})$ 
21:    $AP_{l+1} := \{\}$ 
22:   for entry  $\in V_{G,l}$  with key  $p$  do
23:     if  $(V_{G,l}[p] \geq \tau)$  then  $AP_{l+1} := AP_{l+1} \cup \{(p||0), \dots, (p||b-1)\}$ 
24:    $IL := \{\}$ 
25:   for entry  $\in V_{G,d}$  with key  $p$  do
26:     if  $(V_{G,d}[p] \geq \tau)$  then  $IL := IL \cup \{(p||0), \dots, (p||b-1)\}$ 
27: Output  $IL$ 

```

■ **Figure 2** Computing the inclusion list. Functionalities $\mathcal{F}_{\text{sum}}$ and $\mathcal{F}_{\text{MPC}}$ are specified in Figure 3.

4.2 Desired functionality

Formally, we logically map all possible transactions, whether or not the transaction really exists, into the exponentially large output space of a collision resistant hash function H . Each party's input to our protocol will be an exponentially large vector with 1s in indices corresponding to hashes of transactions. When an inclusion list committee member holds a list of transaction hashes $L' = (\text{tx}_1, \dots, \text{tx}_m)$ that it wants in the list, it logically transforms this list into a bit vector L of length 2^κ where an entry i holds 1 if there exists $\text{tx}_j \in L', j \in [m]$, such that $H(\text{tx}_j) = i$. All other entries are set to 0. Importantly, this exponentially long vector is a logical construct and never concretely realized.

We will use a *private sum* protocol for n committee members to go from $L_1, \dots, L_n$ to a single list $L = \Sigma_{i=1}^n L_i$. Entries of L that are greater than some threshold τ will be added to the final inclusion list. Importantly, the protocol will allow the committee members to efficiently recover only the non-zero entries of the sum vector L , thereby doing work at most linear in the size of the inclusion list and logarithmic in the output space of the hash function to recover the list, rather than the exponential work of a naïve approach.

4.3 Construction details

Our high-level approach is inspired by work on private computation of *heavy hitters* [6] in data held secret shared across a number of parties. Our protocol will allow committee members to recursively explore levels of a binary tree with 2^κ leaves, where each internal

$\mathcal{F}_{\text{MPC}}$: level- l MPC aggregation

```

1: for  $i \in \{0, \dots, n-1\}$  do
2:   for each entry with key  $p$  in  $V_{i,l}$  do
3:     if  $V_{i,l}[p] \cdot (1 - V_{i,l}[p]) \neq 0$  then
4:       Abort/ignore input
5: for each entry with key  $p$  in  $V_{-,l}$  do
6:    $V_{G,l}[p] := \sum_{i=0}^{n-1} V_{i,l}[p]$ 
7: Return  $V_{G,l}$ 

```

 $\mathcal{F}_{\text{sum}}$: coin-tossing

```

1: for  $i \in \{0, \dots, n-1\}$  do
2:   for  $j \in \{0, \dots, \lambda-1\}$  do
3:     if  $\text{coin}_i[j] \cdot (1 - \text{coin}_i[j]) \neq 0$ 
4:       then Abort/ignore input
5: Return  $\sum_{i=0}^{n-1} \text{coin}_i$ 

```

■ **Figure 3** $\mathcal{F}_{\text{MPC}}$ is the level- l MPC aggregation algorithm, and $\mathcal{F}_{\text{sum}}$ is the coin-tossing algorithm. All sums are component-wise. A secure-with-abort MPC aborts on failed bit checks, whereas a robust MPC discards invalid inputs.

node consists of an indicator bit revealing whether it is on the path to a non-zero leaf, and the non-zero leaves make up the elements of the list L . Our exploration algorithm will follow only the paths that lead to non-zero entries in L , without revealing any individual member's contribution.

At each level of the tree, committee members compute the sum of their values for each tree node using a simple MPC protocol that only consists of additions. The sums are revealed without anyone learning which members contributed a zero or one for each node. Once the sums are public, the members assemble a list of those nodes whose counts exceed the committee's chosen threshold τ for inclusion in the final list. These are the only nodes that lie on paths which could possibly have a non-zero value in their leaves. The process is repeated for the children of each node whose count is above the threshold. At the κ^{th} level of recursion, the remaining nodes are in fact the non-zero entries of the list L , computed without revealing who contributed to each entry.

4.4 Protocol description

The resulting protocol is described in Figure 2; we present a more general protocol with branching factor b . For now, we ignore lines 8-12. The protocol runs for $\log_b 2^\kappa$ iterations (line 15). In the first iteration, the protocol considers b number of prefixes; thus representing a $\log_2 b$ prefix length of transaction hashes. In each iteration, we increase the prefix length by $\log_2 b$. A party P_i maintains a list of active prefixes AP_l to be considered for level l (line 13). Based on the active prefix AP_l , P_i creates a local list $V_{i,l}$ where each element of a list is a key-value pair with a prefix $p \in AP_l$ being the key and a boolean value indicating whether it has an input transaction $tx \in L_i$ with prefix p (lines 17-19). Thus, a value 1 indicates that the party is interested in exploring the path with prefix p based on its input L_i , and it will be explored if sufficiently many other parties also input 1 for the same prefix. To determine this, the parties engage in a multiparty computation protocol $\mathcal{F}_{\text{MPC}}$ (line 20) where party P_i inputs $V_{i,l}$ (described in Figure 3). A secure multiparty computation protocol ensures that the parties only learn the output of the function computed on the parties' inputs, and nothing else; thus, a party's interest in exploring a path based on its input is kept private. In a semi-honest version of this MPC protocol, $\mathcal{F}_{\text{MPC}}$ only adds input $V_{i,l}[p]$ for each prefix p to produce output $V_{G,l}$ (Figure 3 lines 5-6). Party P_i then processes the output $V_{G,l}$ to determine the set of prefixes p that appears $\geq \tau$ times; if a prefix p matches that criteria, we explore prefixes $(p||0), \dots, (p||b-1)$ in the next iteration (line 22). At the end of the $\log_b 2^\kappa$ -th iteration, $AP_{\log_b 2^\kappa - 1}$ stores the desired set of outputs.

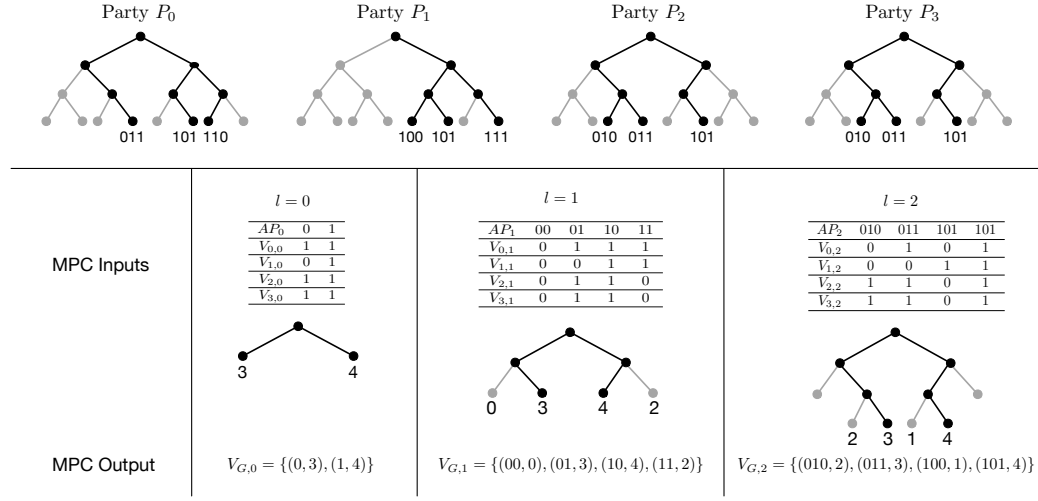


Figure 4 An example execution. For parameters $n = 4$ parties, $\kappa = 3$, $|L_i| = 3$, $b = 2$, and $\tau = 3$. For ease of exposition, we assume all parties have coin values set to 1. At the top, we show the inputs of each party P_i – the black paths denote the paths to input transactions, and the remaining paths are greyed out. In the first iteration, there are only two active prefixes $AP_0 = \{0, 1\}$. Parties input a 1 to a path they are interested in exploring. All parties are interested in exploring both paths except party P_1 , which does not want to explore path 0 since none of its inputs start with that prefix. At the end of the computation, parties output $V_{G,0}$ with 3 parties voting to explore prefix 0 and 4 parties voting to explore prefix 1. Since both these values exceed threshold τ , all four paths are explored in the next iteration $l = 1$. Similarly, at the end of this iteration, only two paths 01 and 10 stay active. Finally, at the end of the execution, the parties output 011 and 101 as the output transactions.

4.4.1 Parameter tuning for security and performance

The inclusion threshold τ should be set based on expectations of the number of Byzantine/-malicious parties against which the committee expects to protect itself. For example, if there is an expectation of t Byzantine parties, the threshold can be set to $\tau = t + 1$ so that the Byzantine parties cannot corrupt or disrupt the output of the protocol, e.g., by not providing data corresponding to the transaction.

At any level of recursion, the number of subprotocols in progress is upper-bounded by the total number of items that need to be put in the final inclusion list, within a factor of b . Thus, increasing the value of b increases the number of prefixes processed per level, but the number of levels reduces to $\log_b 2^\kappa$. Since κ affects only the correctness of the protocol and not its security, we can set κ smaller to speed up the protocol, balancing the probability of collisions where it is unclear which of two or more potential transactions are intended to be on the list. These collisions are not fatal to the protocol because if an unnecessary transaction is added, the correct one can be added in the next iteration of the inclusion list protocol. Figure 4 illustrates an execution of the semi-honest example of our protocol.

4.5 Ensuring Plausible Deniability

The scheme described in Section 4.1 satisfies our anonymity requirement (assuming parties do not deviate from the protocol), but it does not provide plausible deniability. If every member of the inclusion list committee includes a given transaction, revealing the sum eliminates all parties' plausible deniability. If there are n parties and we see that n of them included a transaction, it is immediately clear that all n parties included that transaction.

More precisely, at each level of our protocol, $\mathcal{F}_{\text{MPC}}$ reveals the frequency with which a given prefix appears (line 20), allowing us to perform a cleartext comparison with threshold τ (line 23). This allows an observer to discern whether everyone on the committee included a transaction with that prefix. Plausible deniability against an adversary who controls t committee members can be compromised by a sum as low as $n - t$ since the adversary knows the votes of the t parties it controls.

From a privacy standpoint, this problem can be easily addressed by performing the required comparison with τ within $\mathcal{F}_{\text{MPC}}$; however, computing a comparison inside an MPC incurs additional multiplication gates, increasing the communication, round, and time complexity of each level of the protocol. Instead, we add plausible deniability by adding in a lightweight, randomized mechanism whereby certain parties, chosen privately and at random, always contribute zeros to the counts. Since any party can claim to be among the ones who only contributed zeros, plausible deniability is retained.

As mentioned above, the plausible deniability issue only occurs if the revealed count for a prefix is $\geq n - t$ since $\leq t$ of the inputs are adversarial and the adversary may input 0; thus, if we can guarantee that the output is always $\leq n - (t + 1)$, then an honest party has plausible deniability. We achieve this by ensuring that at least $t + 1$ parties are secretly designated to input all 0s for all levels of the protocol; we will call them *zero-input parties*. Since t is the upper bound on the number of adversarial parties, there must be at least one honest zero-input party. Since the identity of the zero-input parties remains secret, any party can claim to have been one of them, maintaining plausible deniability.

4.5.1 Selecting zero-input parties

Our intuitive idea is for each party to privately toss a 0/1 coin at the beginning of the protocol. Any party whose coin comes up 0 acts as a zero-input party. Unfortunately, this intuition does not work exactly because there is a possibility of there being too few or too many zero-input parties selected, which would result in either plausible deniability being lost or the protocol failing to produce any outputs, respectively. To remedy this, the parties reuse our private sum functionality to ascertain the number of zero-input parties, and they retoss their coins if there are too many or too few parties selected. To avoid extra communication rounds, we have the parties toss and sum many coins in parallel and use the first successful result.

The precise logic for selecting zero-input parties appears in lines 8-12 of Figure 2. The minimum acceptable sum of coin tosses is set to $n - x$, where x is the maximum permissible number of zero-input parties, where any larger number of zero-input parties could prevent honest parties from being able to include transactions on the inclusion list. The parameter x provides a trade-off between plausible deniability and the number of honest parties that must input a transaction for it to be guaranteed in the output. We discuss how to determine the range of acceptable values of x in terms of other parameters in the full version [32].

4.6 Protecting Against Malicious Behavior

In this section, we discuss our protocol in the malicious setting where the adversarial parties may deviate arbitrarily from the prescribed protocol. Recall that our goal is to obtain correctness, soundness, and privacy for the honest parties in the presence of such an adversary.

4.6.1 Using MPC secure under malicious adversaries

First, we ensure that the underlying $\mathcal{F}_{\text{MPC}}$ functionality described in Figure 3 is implemented under an MPC protocol tolerating t *malicious* adversaries. As described in Section 3.3, our implementation instantiates the protocol with two types of malicious security: malicious security with abort and robust MPC. Running the robust and non-robust MPC protocols in parallel means that a block can either immediately have its inclusion list (the optimistic case) or that an inclusion list is guaranteed to become available for a future block (the pessimistic case).

4.6.2 Addressing adversarial inputs

Our protocol is made up of a number of invocations of a smaller MPC protocol. While malicious security for the MPC ensures that the MPC functionalities are computed correctly, a malicious party can still attempt to break security by manipulating its inputs to the MPC (sub-)protocols. Observe that a malicious adversary in an MPC protocol has input independence, i.e., it can input anything to the protocol. By doing so, the adversary can potentially attempt to disrupt the correctness or privacy of the computation. There are two key concerns:

1. **Incorrectness through malformed inputs.** A malicious party may input arbitrary values that interfere with the correct aggregation of votes, potentially causing transactions that should have been considered to be excluded. For example, an attacker could artificially increase the count for a particular node in the tree in order to include an extra transaction in the final list, or even worse, an attacker could send values that “cancel out” the counts sent by other committee members, thereby preventing transactions from making it onto the final inclusion list.
2. **Inconsistent inputs.** The adversary can send inconsistent values across different levels of the protocol. As an example, consider the case where a party votes 1 at level l for a given active prefix $AP_l[p]$, thereby claiming to have transactions matching that prefix. The same party may then later vote 0 at all children vertices of this prefix, which is inconsistent with its earlier input and violates the intended protocol semantics. This can potentially cause additional paths to be explored due to “fake” inputs while honest execution prunes branches that do not meet the voting threshold or vice versa. A malicious party that repeatedly votes “1” for all active prefixes can force the exploration of many unnecessary paths.

For the first case, we enforce input validity through a bit-checking mechanism. Each party is restricted to submitting only binary values $\{0, 1\}$ to indicate its interest in extending a prefix. If a party attempts to inject a non-binary value, the check detects the deviation and causes the computation to abort. Concretely, for each prefix p in party i ’s input $V_{i,l}$ at level l , we ensure $V_{i,l}[p] \cdot (1 - V_{i,l}[p]) = 0$ (Figure 3 (left), lines 1-4). We emphasize that this check is performed over secret-shared values, ensuring that no information about honest inputs is leaked. A similar check is also performed to check for coin values when determining zero-input parties (Figure 3 (right) lines 1-4).

For the second case, note that this behavior does not compromise security guarantees. Since at most t parties are malicious, each can contribute at most one vote per transaction. Thus, a spurious transaction proposed only by malicious parties cannot reach the inclusion threshold, and will never appear in the final list. The effect is limited to performance: In the worst case, the number of paths explored is bounded by $O(|L|) \times b$.

4.6.3 Security analysis

Our complete construction is described in Figures 2 and 3. We analyze the security of the construction and give the proof intuition for the following theorems in the full version [32]. We omit formal cryptographic proofs of the theorems, which would primarily involve invoking the security of the underlying MPC protocols.

► **Theorem 1.** *Assuming it is instantiated with an MPC that satisfies malicious security with abort and parameters are set such that $n - x \geq \tau + t$, $x \geq t + 1$, and $y \geq \tau + x$, our scheme satisfies optimistic correctness with all but negligible probability in the choice of parameter λ .*

► **Theorem 2.** *Assuming it is instantiated with an MPC that provides robustness and parameters are set such that $n - x \geq \tau + t$, $x \geq t + 1$, and $y \geq \tau + x$, our scheme satisfies correctness with all but negligible probability in the choice of parameter λ .*

► **Theorem 3.** *Assuming the malicious security with abort of the underlying MPC protocol, and that $\tau > t$, our inclusion list protocol satisfies soundness.*

► **Theorem 4.** *Assuming the malicious security with abort of the underlying MPC protocol, our inclusion list protocol satisfies anonymity.*

► **Theorem 5.** *Assuming the malicious security with abort of the underlying MPC protocol and that $n - (t + 1) \geq 0$, our inclusion list protocol satisfies plausible deniability.*

5 Implementation and Optimizations

We implemented our malicious-secure privacy-preserving inclusion list protocol using the MASCOT protocol [23] as implemented in the MP-SPDZ [22] MPC framework, and we use HoneyBadgerMPC [26] (referred to as HB-MPC in our evaluation) for the robust variant of the protocol. Our code for both MP-SPDZ and HB-MPC is publicly available ¹.

The rest of this section describes performance-related design decisions and optimizations that improve performance over a straightforward implementation of the scheme as described in the preceding sections. Our implementation includes the first of these optimizations but omits the others, as most of them trade off some security in exchange for improved performance. We evaluated our protocol without these optimizations in order to show performance with the highest security level, but we present them here as they may provide favorable tradeoffs for deployments of this protocol.

5.1 Interacting with mp-spdz

The mp-spdz library incurs some overhead to setup a new protocol each time an MPC starts. This means that running a separate MPC protocol per level of our protocol would result in repeatedly incurring the setup cost. Instead, we run a single MPC protocol that takes additional inputs and returns outputs at each level. However, mp-spdz only allows inputs to MPC protocols from static input files or command-line input. Thus we use the python pexpect module to send and receive inputs to the MPC protocol on a simulated command line rather than using file I/O for inputs.

¹ <https://github.com/timetzw/Privacy-Preserving-Inclusion-List>

This implementation decision comes with a tradeoff, as the library needs to know how many inputs to take at compile time (python-like MPC code is compiled into a final protocol). This means the protocol will need to take in a number of inputs corresponding to the maximum expected inclusion list size at each level, discarding inputs that end up not being needed. We find that this cost is preferable to that of paying the protocol initialization cost at each level.

5.2 Adding concurrency on top of HoneyBadgerMPC

We find that HoneyBadgerMPC runs in a single-threaded mode by default, relying on batching operations rather than concurrency in order to improve performance. While we ran our evaluation in a setting with only one core available per party, we briefly describe how to modify our implementation in order to take advantage of parallelism without altering the protocol. To do this, we could implement our protocol to run in separate MPC processes, each running on a different core. Consider the case of exploring a tree with branching factor four. We can think of each of the four nodes in the first level of the protocol as themselves being the root of a smaller instance of the same protocol, and run each one as a separate process. Since the outcomes of separate paths down a tree are independent of each other, this has no impact on the computation or results of the protocol, except to allow different parts of the protocol to run in parallel on different cores. The same idea applies for any branching factor, so we can split the protocol into as many subproblems as we need to optimally use the computing resources available. Moreover, the branching factor at this first level need not match the branching factor used within each instance of the protocol, allowing for even greater flexibility.

5.3 Speculative execution of subprotocols

We can reduce the overall number of protocol rounds considerably via a speculative subprotocol execution optimization. Since checking that all the sharings consist of 0/1 values requires a round of communication to compute the necessary multiplication, each subprotocol now requires an additional communication round to complete this step. However, the output sums of each subprotocol can be computed before these checks are completed. As such, the next subprotocol can proceed assuming that the checks will work out. In the case that they do, the overall rounds of the protocol are reduced, and in the case that they do not, the additional speculative round will not reveal anything about honest users' inputs because of the security properties of the MPC. Thus our protocol will at all times have a pipeline of subprotocols in progress.

This optimization comes with a mild security tradeoff. Since subprotocols execute speculatively without waiting for all the bit checks of a previous level to complete, it is possible for a malicious party to temporarily inflate/deflate counts for a node in the tree by sending inputs that are not zero or one. While this will eventually be caught, in the meantime, the protocol may explore parts of the tree that do not have sufficient votes to be included, thereby revealing additional inclusion counts that would not have been visible in the honest protocol (while still not revealing which parties are responsible for those counts). This may or may not be a desirable tradeoff for a given deployment scenario.

5.4 Reducing rounds with short trees

We can reduce the round complexity of our protocol by breaking the output of the hash function H into two halves of length $\kappa/2$ and running the protocol separately on each half in parallel. Then a transaction is only included in the inclusion list if both halves of its hash

are present in the protocol output. This increases collision probability but cuts the number of rounds needed in half because the number of protocol rounds depends directly on the length of the hash, which determines the number of protocol levels.

Concretely, the probability of a collision in a list of q transactions is upper bounded by $q^2/2^\kappa$ for a κ bit hash. Since collisions in the left and right halves of a hash function modeled as a random oracle [16, 4] occur independently, collisions will occur with probability at most $\frac{q^2}{2^{\kappa/2}} \cdot \frac{q^2}{2^{\kappa/2}} = \frac{q^4}{2^\kappa}$.

To make the reduction in collision probability less severe, we instead split the output of H into three overlapping bit strings of length $\kappa/2$, requiring a transaction be included in all three halves in order to be included in the list. That is, we run the protocol with the first and second halves of the output of H , but also with the bits in positions $\kappa/4$ to $3\kappa/4$. This increases the computation per round but allows using a smaller value of κ to get the same concrete security level.

6 Evaluation

This section reports on our evaluation of our protocol implementation and empirical comparison to various baselines. See Section 7 for further discussion and qualitative comparison to other approaches which do not have a baseline implementation available.

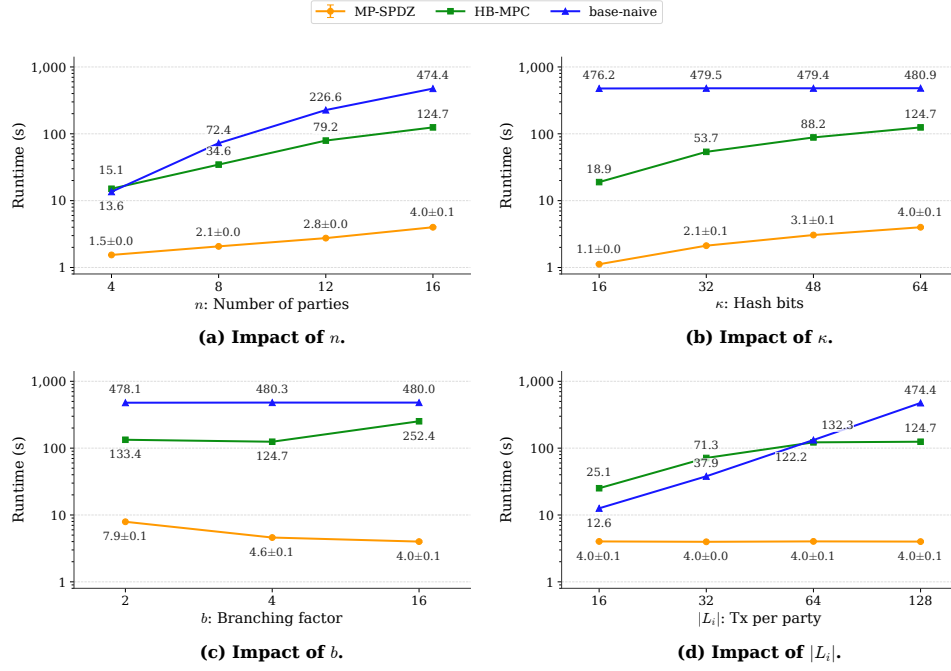
6.1 Experimental setup

We conducted our experiments on four machines connected on a LAN. Each machine is equipped with an Intel Xeon Gold 6248R CPU @ 3.00GHz, running on VMware virtualization. Each of the machines have 16 virtual CPUs, 10Gbps bandwidth, and 256 GB of disk storage. One of the machines has 64 GB RAM whereas the rest have 48 GB RAM.

We note that based on the functionality provided by the underlying framework, the experiments on MP-SPDZ utilized all four machines equally, simulating multiple parties per machine when needed, whereas the ones on HB-MPC were run on a single machine. Moreover, for MP-SPDZ, each party's instance could utilize multiple CPU cores, when available, to improve performance.

6.2 Baselines

A natural point of comparison for our protocol would be other proposals for privacy-preserving inclusion lists. Unfortunately, the existing proposals are sketches and not yet fully formalized, and therefore do not have reference implementations we could use as a baseline. We qualitatively compare our approach to them in Section 7. In the absence of available implementations of other baselines, we consider two different MPC-based baseline algorithms to which we compare our solution. We implement both baselines using MP-SPDZ. The first baseline, **base-naïve**, is a naïve baseline where, for every input transaction from every party, we compare it with every other transaction to check if it appears at least τ number of times. Performing this computation requires $O(n^2|L_i|^2)$ comparisons, where each comparison requires at least $O(\kappa)$ multiplication gates under MPC, where κ is a security parameter corresponding to the length of transaction hashes, even if we ignore the cost of checking for the threshold to reach τ . Observe that implementing the output of **base-naïve** under MPC requires either using an oblivious list or naively adding every element to the secret-shared output and securely shuffling the output before revealing it. Our implementation does not perform the shuffling step, and thus the reported performance is a lower bound on the actual performance cost incurred by this algorithm.



■ **Figure 5** Performance as a function of parameters.

6.3 Experiment parameters and metrics

We choose our parameters in line with EIP-7805 [30], which considers the proposed implementation for inclusion list design (without privacy) for Ethereum. Thus, we consider $n = 16$. We set the mempool size to 128 and let the parties choose transactions from this mempool up to the maximum of $|L_i| = 128$ for all parties P_i . We set $\kappa = 64$; thus, we consider the first 64 bits of a transaction hash in our tree exploration. Moreover, we choose $b = 16$ empirically as the default parameter for MP-SPDZ and $b = 4$ for HB-MPC since it yields better performance (as will be shown later).

The primary metric of interest is the latency for running the inclusion list protocol; thus we report the maximum time across all honest parties for the online portion of the MPC protocol. For each parameter setting, for MP-SPDZ, we repeated the experiments five times and report the mean latency (across all parties and all runs) together with 95% confidence intervals. For base-naïve and HB-MPC, since they have a longer running time, we executed these experiments only once; we present the mean latency across all parties.

6.4 Evaluation Results

6.4.1 Efficiency in the default setting

In the default setting, the MP-SPDZ implementation runs in 4.0 ± 0.1 s whereas the HB-MPC implementation runs in 124.7s. Thus, the execution time for achieving optimistic correctness is within the 12s slot time for Ethereum. The offline time needed for MPC setup under the default setting is 14.7s for MP-SPDZ and 33.2s for HB-MPC. We note that this offline setup can be performed in advance and does not affect the online MPC latency reported above.

The running time of the more efficient MP-SPDZ implementation is dominated by 16,384 MPC multiplications distributed over 32 sequential levels of execution. These results suggest that the core computation performed by our protocol is lightweight enough for slot-level

optimistic execution in our default experimental setting. In this sense, the evaluation supports the computational efficiency of our design, but the number of sequential communication rounds induced by the tree traversal is now the bottleneck.

We also conducted preliminary experiments in a WAN setting. As expected, WAN latency increases the running time of the current implementation. We therefore view the WAN results as a stress test of the current implementation rather than a fully optimized deployment benchmark. Several optimizations described in Section 5, including speculative execution and shorter trees, are designed to reduce the effective round complexity of the protocol. In addition, the tree structure exposes further opportunities for parallelism across independent subtrees. We leave a full WAN-optimized implementation and evaluation of these optimizations to future work.

While our robust protocol variant guarantees output delivery, its latency significantly exceeds the 12-second Ethereum slot time. This reflects a tradeoff between performance and robustness; the HoneyBadgerMPC framework uses heavier tools to provide the stronger robustness property. Thus, while our protocol demonstrates the feasibility of a robust private inclusion list protocol, which could perhaps be run in parallel with the faster, non-robust protocol as a fail-safe, further performance improvements will be needed to meet real-world requirements. Fortunately, improvements in the underlying robust MPC will directly lead to improvements in our protocol’s performance.

6.4.2 Protocol performance as a function of parameters

Figures 5 present the performance of MP-SPDZ and HB-MPC as a function of n , κ , τ , and $|L_i|$.

First, we observe that as n increases, the execution time for both protocols increases. For MP-SPDZ, the multiplication gates scale linearly in n , while communication grows quadratically. For HB-MPC, the communication grows super-linearly and the computation increases linearly in n .

Second, as κ increases, we observe that the execution time increases roughly linearly. Every level of the tree, except for the first few levels, performs a roughly similar amount of computation corresponding to the max number of prefixes/transactions processed per level. In the first few levels, the number of prefixes processed per level increases until it reaches that threshold. Thus, we observe a roughly linear change in latency with linear change in κ .

Third, $b = 16$ works best for MP-SPDZ, and $b = 4$ performs better for HB-MPC. Observe that a larger b implies checking for a larger number of prefixes at once; thus, this increases the amount of communication. At the same time, for a fixed κ , the height of the tree is $\log_b \kappa$, and thus, the number of levels decreases. It is not always guaranteed a monotonic relationship between branching factor and performance.

Finally, the number of transactions $|L_i|$ in each party’s input does not affect performance very much for MP-SPDZ. This is due to a framework constraint where the input size processed at each level must be hard-coded in advance, we therefore set this size to 128, which matches the mempool size used in Section 5. HB-MPC performs faster with smaller size of inputs due to fewer computation counts.

6.4.3 Comparison with baselines

Figure 5 compares the performance of MP-SPDZ with baselines **base-naïve**. We execute the **base-naïve** protocol only for a subset of data points since its performance is much slower than MP-SPDZ. Under the default parameterization, it is $\sim 120\times$ slower than MP-SPDZ.

Our performance improvement over the baselines is explained by the smaller number of MPC multiplications needed in our protocol. Since multiplications of secret shared data are the most expensive operation in an MPC, we seek to minimize the number of multiplications, even at the cost of more additions, which are effectively free. Thus, since our solution generally requires fewer multiplications than **base-naïve**, we have significantly better performance.

7 Discussion

This section discusses general issues related to inclusion lists that are not specific to our protocol. It also compares our approach qualitatively to other proposed or potential approaches to private inclusion lists. Given that there are not concrete proposals or implementations to which we can compare, an empirical comparison is unfortunately impossible. Nonetheless, we hope this discussion helps to frame the tradeoffs results in the context of other related approaches.

7.1 Committee Membership Risks

Our protocol has been designed to be compatible with a number of different deployment scenarios for inclusion lists. For example, committee members can be chosen in any way as long as the selection process ends by publicly identifying the committee members. However, there are certain inherent limitations to this kind of inclusion list design. While correcting the general (non-privacy-related) flaws of inclusion lists is outside the scope of this work, the widespread attention inclusion lists have received makes them an important topic for study.

While private inclusion lists help hide which members of a committee chose to include a transaction in an inclusion list, the fact that a transaction was included means that each member may be subject to heightened scrutiny or suspicion for including that transaction. This places a limit on how much plausible deniability such a protocol can provide, as there is always a τ/n chance that each member included each transaction, by definition of the inclusion criteria. It is possible that this inherent limit on plausible deniability means that the entire committee can be held responsible for including a transaction. The only alternative we foresee to address this limitation is if the committee consists of all validators as in zkFOCIL [2] (discussed in the next subsection).

7.2 Ethereum Proposals

There are two high-level proposals in the Ethereum community. The first is an “anonymous inclusion list” design that combines a committee-based design (akin to this work) with linkable ring signatures [21, 5] to achieve anonymity [1]. All committee members sign their local inclusion lists without revealing their specific identity, instead proving they belong to the valid group. It relies on an implementation of an anonymous broadcast protocol based on Dandelion++ [15] to protect anonymity at the network layer. As described in the blog post [1], a thorough security analysis is necessary to assess the trade-offs between network anonymity and security. Our proposal eliminates the need for linkable ring signatures and replaces the anonymous broadcast with limited use of MPC, whose provable security properties are more easily analyzed and understood.

The second design sketch, called zkFOCIL [2], uses a private lottery scheme to select the inclusion list committee obliviously. Their instantiation also relies on linkable ring signatures, and the proposal includes a proposed scheme based on general-purpose SNARKs. This proposal increases the size of the anonymity set for committee members by making members

anonymous among the somewhat larger set of validators from which the committee is chosen, attempting to mitigate the plausible deniability issues inherent to the committee-based IL design. However, anonymity in this approach still relies on the underlying anonymous broadcast mechanism. A recent implementation of zkFOCIL [25] focuses primarily on the timing of private committee selection and proof generation. It does not yet provide an end-to-end implementation and benchmark of the anonymous broadcast channel, so the full online latency of constructing and publishing an inclusion list remains unclear. Moreover, private committee selection provides an additional layer of pseudonymity, but pseudonymity alone does not imply transaction-level plausible deniability: if messages, timing, or dissemination patterns can be linked to the same anonymous participant, then access-pattern-style analysis may still leak additional information. In contrast, our MPC-based design focuses on the transaction-selection layer: given a committee and its members' local transaction-hash lists, we provide and evaluate a concrete protocol for producing an inclusion list while hiding which members contributed to each transaction.

7.3 Other Potential Approaches

Since our protocol provides an efficient way for committee members to anonymously send each other the contents of their inclusion lists, one alternative would be to rely on a classic approach like DC-nets [9], whereby committee members could anonymously broadcast the contents of their inclusion lists and then count up the number of times each transaction has appeared. In a semihonest setting, this approach would require more communication than our scheme, but would otherwise be feasible. However, when considering malicious clients, it becomes more challenging. Malicious clients may try to send random messages to clobber others' contributions or send the same transaction(s) multiple times to inflate their count. While techniques exist to ensure that participants in a DC-net cannot corrupt the outputs of the protocol [20, 10, 11], additional ideas would be needed to ensure that parties cannot enter duplicate inputs.

Another approach to inclusion list construction would be to rely on protocols for private set intersection [18]. While two-party PSI protocols have become increasingly efficient, multiparty protocols lag behind in performance [34]. One of our naïve baselines was to build a PSI-type protocol on top of MPC, and perhaps this approach could be refined to build more efficient protocols. Specifically, inclusion list formation as described here can be thought of as a multi-party PSI cardinality problem. In fact, there is such a protocol implicit in our construction, albeit with additional leakage that would not be suitable for the standard PSI problem formulation. Moreover, just as our core semihonest protocol needs additional protections for malicious behavior and to preserve deniability, any PSI protocol would need to be modified to provide similar features as well.

8 Conclusion

We have presented a new protocol to achieve privacy-preserving inclusion lists. Our protocol allows members of an inclusion list committee to collectively construct an inclusion list without revealing which transactions they have chosen to include and while maintaining plausible deniability that they did not contribute any particular transaction. In the presence of centralization of block proposing power, our hope is that our protocol can allow committee members to prevent retaliation and ensure robust censorship resistance.

References

- 1 0xF1D. Anonymous inclusion lists (anon ils). <https://ethresear.ch/t/anonymous-inclusion-lists-anon-ils/19627>, 2024. Accessed: 2025-08-21.
- 2 0xF1E. zkfocil: Inclusion list privacy using linkable ring signatures. <https://ethresear.ch/t/zkfocil-inclusion-list-privacy-using-linkable-ring-signatures/21688>, 2025. Accessed: 2025-08-21.
- 3 Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.
- 4 Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, 1993.
- 5 Ward Beullens, Shuichi Katsumata, and Federico Pintore. Calamari and falafel: logarithmic (linkable) ring signatures from isogenies and lattices. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 464–492. Springer, 2020. doi:10.1007/978-3-030-64834-3_16.
- 6 Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 762–776. IEEE, 2021. doi:10.1109/SP40001.2021.00048.
- 7 Vitalik Buterin. State of research: increasing censorship resistance of transactions under proposer/builder separation (PBS). HackMD, January 2022. URL: https://notes.ethereum.org/@vbuterin/pbs_censorship_resistance.
- 8 Vitalik Buterin and Mike Neuder. No free lunch – a new inclusion list design. Ethereum Research, August 2023. URL: <https://ethresear.ch/t/no-free-lunch-a-new-inclusion-list-design/16389>.
- 9 David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptology*, 1(1):65–75, 1988. doi:10.1007/BF00206326.
- 10 Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 340–350. ACM, 2010. doi:10.1145/1866307.1866346.
- 11 Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in verdict. In *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, pages 147–162. USENIX Association, 2013. URL: <https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/corrigan-gibbs>.
- 12 Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012. doi:10.1007/978-3-642-32009-5_38.
- 13 Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two. In *IEEE Symposium on Security and Privacy*, 2018.
- 14 Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. Comprehensive anonymity trilemma: User coordination is not enough. *PoPETs*, 2020(3):356–383, 2020. doi:10.2478/POPETS-2020-0056.
- 15 Giulia Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees. *arXiv preprint arXiv:1805.11060*, 2018. doi:10.48550/arXiv.1805.11060.
- 16 Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

- 17 Elijah Fox, Mallesh M. Pai, and Max Resnick. Censorship Resistance in On-Chain Auctions. In Joseph Bonneau and S. Matthew Weinberg, editors, *5th Conference on Advances in Financial Technologies (AFT 2023)*, volume 282 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.AFT.2023.19.
- 18 Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2004. doi:10.1007/978-3-540-24676-3_1.
- 19 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987.
- 20 Philippe Golle and Ari Juels. Dining cryptographers revisited. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 456–473. Springer, 2004. doi:10.1007/978-3-540-24676-3_27.
- 21 Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 253–280. Springer, 2015. doi:10.1007/978-3-662-46803-6_9.
- 22 Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1575–1590. ACM, 2020. doi:10.1145/3372297.3417872.
- 23 Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 830–842. ACM, 2016. doi:10.1145/2976749.2978357.
- 24 Quintus Kilbourn. Censorship Resistance: crlists in mev-boost issue #215. GitHub, July 2022. URL: <https://github.com/flashbots/mev-boost/issues/215>.
- 25 Shreyas Londhe and Suyash Bagad. zkFOCIL: Implementation & Benchmarking Report - Cryptography, January 2026. Section: Cryptography. URL: <https://ethresear.ch/t/zkfocil-implementation-benchmarking-report/23966>.
- 26 Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. Honeybadgermpc and asynchromix: Practical asynchronous MPC and its application to anonymous communication. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 887–903. ACM, 2019. doi:10.1145/3319535.3354238.
- 27 Credit and debit card market share by network and issuer. <https://www.fool.com/money/research/credit-debit-card-market-share-network-issuer/>, 2025. Accessed: 2025-08-21.
- 28 Mike Neuder and Toni Wahrstätter. Unconditional inclusion lists. Ethereum Research, January 2024. URL: <https://ethresear.ch/t/unconditional-inclusion-lists/18500>.
- 29 Burak Öz, Danning Sui, Thomas Thiery, and Florian Matthes. Who wins ethereum block building auctions and why?, 2024. doi:10.48550/arXiv.2407.13931.
- 30 Thomas Thiery, Francesco D’Amato, Julian Ma, Barnabé Monnot, Terence Tsao, Jacob Kaufmann, and Jihoon Song. EIP-7805: Fork-choice enforced Inclusion Lists (FOCIL). Ethereum Improvement Proposals, July 2024. URL: <https://eips.ethereum.org/EIPS/eip-7805>.
- 31 Thomas Thiery, Barnabé Monnot, Francesco D’Amato, and Julian Ma. Fork-Choice enforced Inclusion Lists (FOCIL): A simple committee-based inclusion list proposal. Ethereum Research, June 2024. URL: <https://ethresear.ch/t/fork-choice-enforced-inclusion-lists-focil-a-simple-committee-based-inclusion-list-proposal/19870>.

- 32 Zhengwei Tong, Saba Eskandarian, and Kartik Nayak. Privacy-Preserving Inclusion Lists, 2026. Publication info: Published elsewhere. Minor revision. To appear in *Advances in Financial Technologies* (AFT 2026). URL: <https://eprint.iacr.org/2026/1582>.
- 33 U.S. Department of the Treasury. OFAC Specially Designated Nationals Data, 2022. URL: <https://www.treasury.gov/ofac/>.
- 34 Jelle Vos, Mauro Conti, and Zekeriya Erkin. Sok: Collusion-resistant multi-party private set intersections in the semi-honest model. In *IEEE Symposium on Security and Privacy, SP 2024*, pages 465–483. IEEE, 2024. doi:10.1109/SP54263.2024.00079.
- 35 Sarisht Wadhwa, Julian Ma, Thomas Thiery, Barnabe Monnot, Luca Zanolini, Fan Zhang, and Kartik Nayak. AUCIL: An inclusion list design for rational parties. *Cryptology ePrint Archive*, Paper 2025/194, 2025. URL: <https://eprint.iacr.org/2025/194>.
- 36 Anton Wahrstätter, Jens Ernstberger, Aviv Yaish, Liyi Zhou, Kaihua Qin, Taro Tsuchiya, Sebastian Steinhorst, Davor Svetinovic, Nicolas Christin, Mikolaj Barczentewicz, and Arthur Gervais. Blockchain Censorship, June 2023. arXiv:2305.18545 [cs]. doi:10.48550/arXiv.2305.18545.
- 37 Fredrik Winzer, Benjamin Herd, and Sebastian Faust. Temporary censorship attacks in the presence of rational miners. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 357–366. IEEE, 2019. doi:10.1109/EUROSPW.2019.00046.
- 38 Fei Wu, Thomas Thiery, Stefanos Leonardos, and Carmine Ventre. From competition to centralization: The oligopoly in ethereum block building auctions, 2024. doi:10.48550/arXiv.2412.18074.
- 39 Sen Yang, Kartik Nayak, and Fan Zhang. Decentralization of ethereum’s builder market, 2025. doi:10.48550/arXiv.2405.01329.