Multiple Concurrent Proposers: Why and How

Pranav Garimidi pgarimidi@a16z.com
Joachim Neu jneu@a16z.com
Max Resnick max.resnick@anza.xyz

Version: ddd6949 @ 2025-09-28

Traditional single-proposer blockchains suffer from miner extractable value (MEV), where validators exploit their serial monopoly on transaction inclusion and ordering to extract rents from users. While there have been many developments at the application layer to reduce the impact of MEV, these approaches largely require auctions as a subcomponent. Running auctions efficiently on chain requires two key properties of the underlying consensus protocol: *selective-censorship resistance* and *hiding*. These properties guarantee that an adversary can neither selectively delay transactions nor see their contents before they are confirmed. We propose a *multiple concurrent proposer* (MCP) protocol offering exactly these properties.

1 Introduction

A fundamental design flaw in modern blockchain protocols is the creation of a serial monopoly over transaction inclusion. In prevailing single-proposer systems, a single node[1] is granted temporary but absolute authority to determine the composition of the next block, allowing it to extract significant economic rents, commonly known as *miner extractable value* (MEV) [32]. This monopolistic power to censor, reorder, and insert transactions creates profound market distortions, undermines the principle of fair access, and severely constrains the set of viable on-chain financial applications. While numerous countermeasures to MEV have been proposed, the underlying cause has not been satisfactorily addressed: a market structure that grants a single proposer monopolistic control over transaction ordering and inclusion.

This paper develops a framework for analyzing these market failures by reducing the broad and amorphous MEV landscape to its economic core. The most persistent and lucrative forms of MEV are: cross-exchange arbitrage, atomic arbitrage, and liquidations.[2] We argue that these activities are best understood through the lens of auction theory; specifically, as high-frequency public-value auctions where a publicly observable signal evolves continuously, creating opportunities for bidders to compete and capture profits.

Today, running an auction that starts and finishes in the span of a single block is essentially impossible, for two reasons. First, in many systems a single node controls which transactions

[1] We use "node" and "validator" interchangeably in this paper.
[2] We note that sandwiching can be solved at the application layer even on single-proposer blockchains [75].

are included in the next slot (which means, in particular, that node controls which auction bids are included) [40]. Second, that same node sees transaction contents before all transactions (and therefore bids) are confirmed (and thus, while the node can still decide what bid of its own to place, if any). We show in Sec. 1.1 that either of these issues makes running an auction on chain non-viable. Addressing this failure requires a protocol that can guarantee both *selective-censorship resistance* and *hiding*. We formally define both of these properties, and show that our *multiple concurrent proposers* (MCP) protocol (Sec. 3) satisfies them, ensuring immediate inclusion for transactions submitted to honest nodes, while concealing their contents until the consensus decision is final. While short-term censorship resistance [3] and hiding have been studied before, to our knowledge, we are the first to define the properties in a protocol-agnostic way, and to devise a protocol satisfying both properties simultaneously.

The importance of censorship resistance and hiding for on-chain market structure has been established before [40, 54]; however, no major blockchain has incorporated these properties into its consensus protocol. A common critique of multiple concurrent proposer designs is that these properties cannot be achieved without introducing significant overhead. Our main contribution is to provide MCP as a light-weight gadget that fits on top of any standard PBFT-style consensus protocol. We prove that this construction is safe, live, and provides both censorship resistance and hiding, with an additional latency overhead of only a single network round-trip after optimizations compared to latency optimal protocols without the economic guarantees.

While many other solutions to MEV have been proposed, none of these approaches are without significant shortcomings (see Sec. 1.2 for a detailed discussion). Some protocols have addressed hiding via encrypted mempools and some protocols have achieved approximations of short-term censorship resistance with DAG-based protocols, but no protocol has achieved the strict version of selective-censorship resistance we describe here. Thus, turning away from addressing MEV directly, protocols have been designed to reduce the negative externalities of MEV such as consensus instability or network centralization [58, 8, 42]. It should be noted that while approaches like proposer-builder separation in Ethereum have worked to reduce the adverse side effects of MEV at the protocol level, they have done so by facilitating MEV at the application layer, degrading the efficiency of on-chain financial applications.

The remainder of this paper proceeds as follows. In Sec. 1.1, we discuss three examples that illustrate the market failures introduced by the leader's monopoly on transaction inclusion/ordering, motivating our definitions of censorship resistance and hiding, before discussing related work in Sec. 1.2. In Sec. 2, we recapitulate cryptographic primitives, and define the network, consensus, and economic models that we rely on throughout the paper. In Sec. 3, we present the MCP protocol in detail. In Sec. 4, we show that the MCP protocol, coupled with a safe and live underlying atomic broadcast primitive, is secure and achieves the desired censorship resistance and hiding properties.

1.1 Three Examples

We present three stylized games for auctions. Our goal here is twofold: first, to provide the reader with an understanding of how specific aspects of an auction's market structure—such as the ability of one bidder to censor other bidders, see their bids, or bid with slightly more up-to-date information—can be more or less harmful to the efficiency of the system; and second, to explicitly state the market structure our protocol aims to achieve.

Three properties of single-proposer blockchains make running an auction in a single slot difficult:
1. The proposer can censor bids.
2. The proposer can see other bids and change his own bid based on that.
3. The proposer has a latency advantage compared to other bidders.

We begin with a scenario where all three effects are present. We then remove these, one by one, to isolate their impact and motivate our desired properties and the design of our protocol.

Two bidders, Alice and Bob, are competing in a first price auction for an item for which both bidders have a common value that continuously changes over time. At time t, the value of the item is observed to be v_t by both bidders. We refer to this value as the signal at time t. The auction concludes at time τ at which point the item is transferred to the winning bidder. We assume the item's value follows a martingale, so that $E[v_\tau|\{v_s \mid s \leq t\}] = v_t$ for all $t \leq \tau$. This means at any given time, both bidders' expected future value of the item at the end of the auction is equal to the latest value they have observed.

We assume that bidders are risk neutral and have quasi-linear utilities, so that Alice's utility is $u_{\mathrm{A}}(b_{\mathrm{A}}, b_{\mathrm{B}}) = \mathbb{1}_{b_{\mathrm{A}} > b_{\mathrm{B}}}(v_\tau - b_{\mathrm{A}})$, with Bob's utility defined analogously. Alice bids first at time t_{A} observing the signal $v_{t_{\mathrm{A}}}$, and Bob bids after at time $t_{\mathrm{B}} > t_{\mathrm{A}}$ observing the updated signal $v_{t_{\mathrm{B}}}$. For tractability throughout, we assume that if Alice and Bob have the same bid, Bob wins the item.

Let us consider, as our first example, a game where Bob has every possible advantage. He bids after Alice, sees her bid, observes new information about the item's value, and can censor her bid entirely.

Scenario 1: Neither censorship resistance nor hiding
1. At time t_{A}, Alice observes the signal $v_{t_{\mathrm{A}}}$ and submits a bid b_{A}.
2. At time t_{B}, Bob observes Alice's bid b_{A}, the new signal $v_{t_{\mathrm{B}}}$, and then chooses whether to censor Alice's bid and submits his own bid b_{B}.
3. If Bob censors, only his bid is considered. If not, both bids are. The bidder with the highest included bid wins, values the item at v_τ, and pays their bid.

In this game, Bob's optimal strategy is trivial and devastating for the auction's fairness and efficiency. He can simply choose to censor Alice's bid, regardless of its value or the new signal $v_{t_{\mathrm{B}}}$, and submit a near-zero bid himself. By doing so, he guarantees a win at virtually no cost. Censorship alone is so powerful that it completely breaks the auction, regardless of whether Bob can see Alice's bid.

In response to the problem of censorship, protocols suggesting multiple concurrent proposers have emerged to mitigate it [40]. Let us assume such a system is implemented, successfully removing Bob's ability to selectively cancel Alice's bid. Is the resulting auction efficient? In the second example, we explore a market where censorship is resolved, but Bob still bids second, sees Alice's bid, and has more recent price information.

Scenario 2: Censorship resistance but no hiding
1. At time t_{A}, Alice observes the signal $v_{t_{\mathrm{A}}}$ and submits a bid b_{A}.
2. At time t_{B}, Bob observes b_{A}, the updated signal $v_{t_{\mathrm{B}}}$, and then places his bid b_{B}.
3. The highest bidder wins, values the item at v_τ and pays their bid.

In this game, Bob has a very simple best response to Alice's bid b_{A} and the new signal $v_{t_{\mathrm{B}}}$:

$$b_{\mathrm{B}}(b_{\mathrm{A}}, v_{t_{\mathrm{B}}}) = \begin{cases} b_{\mathrm{A}}, & v_{t_{\mathrm{B}}} \geq b_{\mathrm{A}} \\ 0 & v_{t_{\mathrm{B}}} < b_{\mathrm{A}} \end{cases} \tag{1}$$

Notice that when Bob plays this strategy, Alice is subject to extreme *adverse selection*. Whenever the price moves favorably, such that she would have had a positive payoff ($v_{t_{\mathrm{B}}} > b_{\mathrm{A}}$), Bob outbids her and captures the surplus. Whenever the price moves unfavorably against her ($v_{t_{\mathrm{B}}} < b_{\mathrm{A}}$), Bob lets her win, forcing her to take the loss. Knowing this, the subgame perfect equilibrium of this game is for Alice to bid nothing at all, allowing Bob to win the item for free. Even without censorship, the side information (Bob seeing b_{A}) is enough for the auction to unravel completely. The acute adverse selection in this second scenario stems from Bob's ability to "peek" at Alice's bid.

As our third and final example, we consider what happens when this ability is removed, leaving only the unavoidable *last look* [54] feature where Bob's only advantage is bidding with a more recent signal than Alice.

Scenario 3: Censorship resistance and hiding

1. At time t_{A}, Alice observes the signal $v_{t_{\mathrm{A}}}$ and submits a bid b_{A}.
2. At time t_{B}, Bob observes the updated signal $v_{t_{\mathrm{B}}}$ (but does not observe b_{A}), and then places his bid b_{B}.
3. The highest bidder wins, values the item at v_τ and pays their bid.

In this game, the equilibrium from the second example, where the auction completely unraveled, is no longer possible. Bob cannot condition his strategy on Alice's bid because he cannot observe it. We note that the equilibrium strategy profile is still hard to determine as any pure strategy played by Alice would allow Bob to know exactly what Alice bid (even without explicitly seeing her bid) causing the same unraveling as the previous case. The exact mixed-strategy equilibrium played will depend on the process by which the asset's price is determined. But, prior work has shown that if the price is determined by a geometric Brownian motion process, then the revenue of the auction converges to $v_{t_{\mathrm{A}}}$ as Bob's time advantage (and thus his informational advantage) tends to zero [54]. This is a notably different, and normatively much more desirable, outcome than the complete unraveling in the earlier second scenario, which occurs even for the smallest information advantage, if Bob can see Alice's bid. The effect of last look, when isolated, is not nearly as detrimental in a system with censorship resistance and hiding, and can be further reduced by increasing the block production frequency.

Now that we have explored these motivating examples—arriving at a functional market structure by peeling away layers of asymmetric advantage—we can proceed with the main body of our work, which is to propose a protocol that provides selective-censorship resistance and hiding, and thus can run an auction implementing Scenario 3 rather than Scenarios 1 or 2.

1.2 Related Work

We start by reviewing earlier MEV mitigation approaches and their challenges.[3] Broadly, there are three approaches to addressing MEV in the literature: fair-ordering consensus protocols [48, 50, 82, 47, 24, 23, 28, 77, 56], transaction encryption [81, 55, 18, 30, 19, 52, 72], and batching [21, 44, 72].

[3]For comprehensive surveys of MEV countermeasures, see [79, 43].

Fair Ordering Fair ordering protocols try to extend consensus to achieve a strict *order fairness* property, where if a majority of nodes observe transaction tx_1 before transaction tx_2 then tx_1 should be sequenced before tx_2 in the final output log. But, with realistic network delays among a globally distributed node set and some nodes acting adversarially, so-called Condorcet cycles arise, where a majority of nodes each may see a transaction tx_1 before transaction tx_2, tx_2 before transaction tx_3, and tx_3 before tx_1. Because of these cycles it is theoretically impossible to achieve the full order fairness property [48]. The Aequitas and Themis protocols [48, 47] achieve a weaker *batch order fairness* property (which is still stronger than the fairness properties provided by Wendy [50] and Pompe [82]), where if "many" nodes have seen transaction tx_1 before transaction tx_2, then tx_2 would at least not be in an earlier consensus batch than tx_1.

Although batch order fairness is one of the strongest definitions of fairness that has been achieved in the literature, its usefulness in practice has remained limited since a motivated attacker can create large cycles by inserting transactions [73], at which point the batch order fairness guarantee becomes weak. Recent developments in fair ordering protocols have mostly been performance improvements [77], not protocols that offer stronger fairness guarantees than Aequitas/Themis. There has also been work on defining order fairness in the universal composability framework [31]. The authors of [31] give lower bounds on the degree to which order fairness can be achieved by any protocol along with a matching upper bound with a protocol using trusted enclaves.

Encrypted Mempools Transaction encryption approaches aim to use cryptography to conceal details of transactions while consensus is underway, so that the proposer and consensus leader have limited information about how to profitably reorder transactions or which transactions to censor to extract value. *Threshold encryption* [18, 17, 10, 81, 16, 30, 2, 14] is the more popular cryptographic primitive chosen for this purpose, but *time-based cryptography* [13] and *identity-based encryption* [55, 36] have also been considered. Encryption-based approaches have several key shortcomings. First, in order to prevent denial-of-service (DoS) attacks, transactions must contain at least some unencrypted metadata about fees and the transaction sender, which presents an attack angle for MEV extraction. Second, since the proposer is still a monopolist on inclusion, they can simply choose only to include transactions that are not encrypted (or which are encrypted but accompanied by the unencrypted transaction through a side channel). Time-sensitive transactions have no choice but to send the unencrypted transaction to the leader, thereby bypassing the mechanism, or wait for the next consensus round (i.e., drop out of on-chain auctions).

Batching Batching approaches aim to remove the power of the proposer to decide the order of transactions in the block, by making execution semantics order-agnostic [44, 60, 61] (or fixing the ordering deterministically, based on the set of transactions included in a block [64, 63]). While batching is a promising way to diffuse the ordering power of the proposer, it does not address the power of the proposer over inclusion and exclusion, and in particular the proposer's power to exclude ("censor") transactions is enough to comprehensively manipulate many of the most popular batch-based mechanisms such as auctions [40]. Another line of work has focused on producing a protocol that has a good market structure for auctions without focusing on integrating it into the broader consensus considerations required on a blockchain [70, 74].

DAG-Based Protocols There is a large body of DAG-based consensus protocols that also allow for multiple nodes to propose transactions in every round. However, these protocols do this to increase throughput rather than for economic concerns. Because of this, design choices are made which make these protocols more performant at the cost of opening up avenues for selective censorship. Many of these protocols (Bullshark [69], Tusk [33], Shoal [68]), only commit special *anchor* blocks, similar to leaders in traditional consensus protocols, in every round and all other blocks are implicitly confirmed based on the blocks an anchor refers to. This gives anchors the power to selectively leave out certain blocks from the previous round. While these blocks may still be picked up by future anchors, this still gives the adversary the power to selectively move certain blocks of transactions to the next round instead of the current round. While not outright censorship, these short-term delays can be enough to be deal-breaking for certain applications, as outlined in Sec. 1.1.

More recent works such as Shoal++ [4] and Sailfish [67] allow for multiple anchor candidates (referred to as leaders in Sailfish) per round to reduce the latency advantage exclusively enjoyed by a single leader node. In theory, this would alleviate the censorship concerns as a single honest anchor in a round would be enough to commit all the honest proposals from the previous round. However, the way both protocols implement multiple anchor candidates still gives an adversary the power to selectively censor. Both protocols have a predetermined order over anchors in each round, and will only commit a prefix of the anchors in this ordering, e.g., if the second anchor in the order crashes, none of the subsequent anchors in that round will be eligible to be committed. The first anchor in this ordering can be thought of as a traditional leader with the subsequent anchors being used to decrease latency. However, in these protocols if the adversary controls the first anchor in a round and crashes the second anchor they can still carry out the same attacks as in the single anchor case.

Additionally, to decrease latency, these protocols are optimistically responsive, and will only wait to hear from $n - f$ nodes before moving to the next round. While this allows these protocols to operate at network speed, it may also result in honest proposals being inadvertently censored. In particular, Mysticeti [6], unlike other protocols, allows blocks to be committed if enough other nodes voted for them, even if the anchor did not vote for them. This means that if a node gets its block to enough honest nodes on time, the anchor has no power to censor it. However, honest nodes only wait to hear from $2f + 1$ other nodes before advancing to the next round, so even honest nodes that announce their blocks on time might still be missed by other honest nodes. If the protocol were changed so that, instead of being optimistically responsive, honest nodes waited until a fixed timeout (guaranteeing that they hear all other honest nodes' proposals) before moving to the next round, then this would not be an issue and the protocol would be selective-censorship resistant. Furthermore, as described, none of these DAG-based protocols satisfy hiding, although hiding can be incorporated into these protocols, for instance using the techniques described in [52].

BigDipper [76], while not DAG-based, is another protocol that allows for multiple nodes to submit transactions for the same round explicitly with the goal of short-term censorship resistance. BigDipper works similarly to our protocol in that it modifies a standard consensus protocol by constraining the types of blocks a leader can propose to including enough mini-blocks in their proposal. However, BigDipper does not use relays like we do, and in turn does not provide hiding and allows for the leader to exclude up to $2f$ mini-blocks, potentially from honest proposers.

"Leaderless" Consensus Protocols There are also so-called "leaderless"[4] protocols like HoneyBadgerBFT [53] and DispersedLedger [78], targeting the asynchronous common subset formulation of consensus, where every node can include transactions in a round. But these protocols still fall short of achieving full selective-censorship resistance. HoneyBadger only requires nodes to confirm $n - f$ proposals before moving to the next round, potentially allowing for f honest proposals to be censored. DispersedLedger improves on this by guaranteeing that every honest proposal will eventually be included, but it achieves this by allowing later rounds to confirm missed blocks from earlier rounds. Thus, while an adversary cannot censor transactions, they can potentially cause short-term confirmation delays for targeted batches. Both these protocols use cryptography to guarantee hiding before blocks are confirmed.

Verifiable Secret Sharing As becomes clear in Sec. 3, our MCP protocol builds on verifiable secret sharing techniques. Secret sharing was first introduced by [65, 12], as a protocol with which a dealer can share a secret with a committee, such that any group of "many" committee members can recover the secret, while any group of "few" committee members learn nothing about the secret. Verifiable secret sharing [29] (VSS) ensures that the secrets recovered by different "large" groups of committee members agree. Packed multi-secret secret sharing [41] allows amortizing communication overhead if the dealer shares multiple secrets simultaneously. In this taxonomy, a component of MCP implements packed VSS. Starting with [38, 59], many VSS schemes were proposed in the past four decades—see [22, 45, 7, 34, 66] and references therein. The closest relative of our MCP protocol among VSS protocols is the acknowledgment-based approach of [35]. Earlier works combining VSS and consensus include [9, 11]. In [9], the focus is on hiding the state (rather than the transactions) of a replicated state machine. No censorship resistance is achieved. Similarly, [11] uses a blockchain as "control plane for a storage system" "of secret information".

2 Preliminaries & Model

2.1 Cryptographic Primitives

Our scheme employs existentially-unforgeable signatures and vector commitments, which we recall below. As is customary in the literature, we consider a *probabilistic polynomial-time* (PPT) adversary $\mathcal{A}$. A function f is *negligible* if for every $c > 0$ there exists an n_0 such that for all $n \geq n_0$, $f(n) < 1/n^c$. We denote $[N] \triangleq \{1, ..., N\}$. We use a computational security parameter κ, and a finite field $\mathbb{F}$ with $|\mathbb{F}| \geq 2^\kappa$.

2.1.1 Signatures

Definition 1 (Signatures)**.** A signature scheme Sig consists of three algorithms:

- **Key generation** $\mathsf{Sig.Gen}\colon 1^\kappa \to (\mathsf{sk}, \mathsf{pk})$**.** Given security parameter κ, the key generation algorithm outputs a *secret key* sk and a corresponding *public key* pk.
- **Signing** $\mathsf{Sig.Sign}\colon (\mathsf{sk}, m) \to \sigma$**.** Given a secret key sk and a *message* m as inputs, the signing algorithm outputs a *signature* σ.

[4] The component protocols of these protocols are still based on leaders.

- **Verification** $\mathsf{Sig.Verify}\colon (\mathsf{pk}, m, \sigma) \to \{0,1\}$**.** Given a public key pk, a message m, and a signature σ as inputs, the verification algorithm outputs 1 if the signature is valid, and 0 otherwise.

A "good" signature scheme satisfies the following two properties:

- **Correctness.** For all key pairs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Sig.Gen}(1^\kappa)$ and all messages m, if $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, m)$, then $\mathsf{Sig.Verify}(\mathsf{pk}, m, \sigma) = 1$.
- **Existential unforgeability.** Every PPT adversary $\mathcal{A}$ succeeds in the following game only with probability negligible in κ: Generate $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Sig.Gen}(1^\kappa)$ and give pk to adversary $\mathcal{A}$. The adversary $\mathcal{A}$ may make polynomially many signing queries m_i and receive signatures $\sigma_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, m_i)$. Finally, $\mathcal{A}$ outputs a forgery (m^*, σ^*). The adversary succeeds iff $\mathsf{Sig.Verify}(\mathsf{pk}, m^*, \sigma^*) = 1$ and m^* was not queried from the signing oracle.

  An example signature scheme satisfying Def. 1 is the BLS signature scheme [15].

2.1.2 Vector Commitments

Definition 2 (Vector Commitments)**.** For a fixed positive integer L, finite set $\mathcal{V}$ of potential *values*, and finite set $\mathcal{R}$ of potential *masking values*, with $|\mathcal{R}| \geq 2^\kappa$ where κ is a security parameter, a vector commitment scheme VC consists of three algorithms:

- **Commitment** $\mathsf{VC.Commit}\colon (\boldsymbol{v}, \boldsymbol{r}) \to C$**.** Given a *vector* $\boldsymbol{v} = (v_1, ..., v_L) \in \mathcal{V}^L$ and *randomness* $\boldsymbol{r} = (r_1, ..., r_L) \in \mathcal{R}^L$ as inputs, the algorithm outputs a deterministic *commitment* C.
- **Opening** $\mathsf{VC.Open}\colon (\boldsymbol{v}, \boldsymbol{r}, i) \to w$**.** Given a vector $\boldsymbol{v} \in \mathcal{V}^L$, randomness $\boldsymbol{r} \in \mathcal{R}^L$, and an *index* $i \in [L]$ as inputs, the algorithm outputs a *witness* w for the value of $\boldsymbol{v}$ at index i.
- **Verification** $\mathsf{VC.Verify}\colon (C, i, v, r, w) \to \{0,1\}$**.** Given a commitment C, an index $i \in [L]$, a claimed *opening* $(v, r) \in \mathcal{V} \times \mathcal{R}$, and a witness w as inputs, the algorithm outputs 1 if the opening is valid, and 0 otherwise.

A "good" vector commitment scheme satisfies the following properties:

- **Correctness.** For all vectors $\boldsymbol{v} = (v_1, ..., v_L) \in \mathcal{V}^L$, all randomness $\boldsymbol{r} = (r_1, ..., r_L) \in \mathcal{R}^L$, and all indices $i \in [L]$: if $C \leftarrow \mathsf{VC.Commit}(\boldsymbol{v}, \boldsymbol{r})$ and $w \leftarrow \mathsf{VC.Open}(\boldsymbol{v}, \boldsymbol{r}, i)$, then $\mathsf{VC.Verify}(C, i, v_i, r_i, w) = 1$.
- **Binding.** Every PPT adversary $\mathcal{A}$ succeeds in the following game only with probability negligible in κ: The adversary $\mathcal{A}$ outputs $(\boldsymbol{v}, \boldsymbol{v}', \boldsymbol{r}, \boldsymbol{r}')$ where $\boldsymbol{v}, \boldsymbol{v}' \in \mathcal{V}^L$ and $\boldsymbol{r}, \boldsymbol{r}' \in \mathcal{R}^L$. The adversary succeeds iff $\boldsymbol{v} \neq \boldsymbol{v}'$ or $\boldsymbol{r} \neq \boldsymbol{r}'$, and $\mathsf{VC.Commit}(\boldsymbol{v}, \boldsymbol{r}) = \mathsf{VC.Commit}(\boldsymbol{v}', \boldsymbol{r}')$.
- **Position binding.** Every PPT adversary $\mathcal{A}$ succeeds in the following game only with probability negligible in κ: The adversary $\mathcal{A}$ outputs $(C, i, v, v', r, r', w, w')$ where $i \in [L]$, $v, v' \in \mathcal{V}$, and $r, r' \in \mathcal{R}$. The adversary succeeds iff $v \neq v'$ or $r \neq r'$, and $\mathsf{VC.Verify}(C, i, v, r, w) = 1$ and $\mathsf{VC.Verify}(C, i, v', r', w') = 1$.
- **Hiding.** Every PPT adversary $\mathcal{A}$ succeeds in the following game only with probability negligible in κ: The adversary $\mathcal{A}$ provides $(\boldsymbol{v}^{(0)}, \boldsymbol{v}^{(1)})$. The challenger flips an unbiased coin $b \xleftarrow{\$} \{0,1\}$, samples randomness $\boldsymbol{r} \xleftarrow{\$} \mathcal{R}^L$ uniformly at random, and gives $C \leftarrow \mathsf{VC.Commit}(\boldsymbol{v}^{(b)}, \boldsymbol{r})$ to $\mathcal{A}$. The adversary $\mathcal{A}$ may make polynomially many opening queries i_j for indices where $v^{(0)}_{i_j} = v^{(1)}_{i_j}$ and receive openings $(r^{(j)}, w^{(j)})$ where $r^{(j)} \leftarrow r_{i_j}$ and $w^{(j)} \leftarrow \mathsf{VC.Open}(\boldsymbol{v}^{(b)}, \boldsymbol{r}, i_j)$. Finally, $\mathcal{A}$ outputs a guess $\hat{b} \in \{0,1\}$. The adversary succeeds iff $\hat{b} = b$.

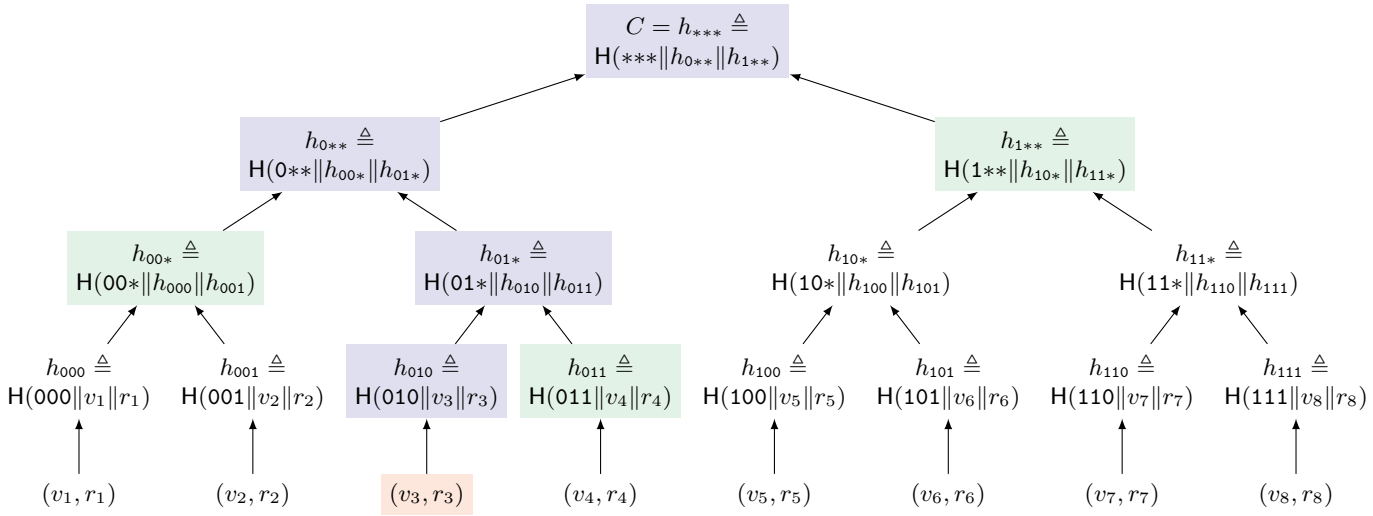

Figure 1: Illustration of hiding Merkle tree vector commitment scheme for $L = 8$. The opening of commitment C at index 3 to (v_3, r_3) (shaded red) is verified by recomputing the hashes (shaded blue) from the opening to the Merkle root using the opening and the Merkle path (shaded green) provided as witness.

To obtain the hiding property of Def. 2, classical (deterministic) Merkle trees [46, Sec. 6.6.2] can be augmented with masking randomness at each leaf. Alg. 1 presents the resulting scheme for vectors of length $L = 2^d$, for some fixed $d \in \mathbb{N}$. Fig. 1 illustrates the construction for $L = 8$. Lengths that are not a power of two can be handled via padding. The construction uses a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$, where κ is the security parameter.

For commitment, $\mathsf{VC.Commit}$ takes a vector $\boldsymbol{v} = (v_1, \ldots, v_L)$ and randomness $\boldsymbol{r} = (r_1, \ldots, r_L)$, and constructs a Merkle tree. Each Merkle tree leaf hash incorporates randomness: $h_{\mathrm{leaf}(i)} = \mathsf{H}(\mathrm{leaf}(i) \| v_i \| r_i)$ for $i \in [L]$ (Alg. 1, ln. 5). The tree builds bottom-up, with hashes at internal nodes computed as $h_b = \mathsf{H}(b \| h_{b_\mathrm{L}} \| h_{b_\mathrm{R}})$ from the hashes of their children nodes h_{b_L} and h_{b_R}, respectively (Alg. 1, ln. 10). The commitment is the root hash. In the example of Fig. 1, the leaf at index 3 (red-shaded) is computed as $h_{\mathtt{010}} = \mathsf{H}(\mathtt{010} \| v_3 \| r_3)$ (note that $\mathtt{010}$ is the $d = 3$-bit binary representation of $3 - 1 = 2$). The tree builds upward with internal nodes (blue-shaded) like $h_{\mathtt{01}*} = \mathsf{H}(\mathtt{01}* \| h_{\mathtt{010}} \| h_{\mathtt{011}})$, $h_{\mathtt{0}**} = \mathsf{H}(\mathtt{0}** \| h_{\mathtt{00}*} \| h_{\mathtt{01}*})$, and $h_{***} = \mathsf{H}(*** \| h_{\mathtt{0}**} \| h_{\mathtt{1}**})$ (note the progressive truncation of the binary hash identifiers), ultimately producing root $C = h_{***}$.

For opening, $\mathsf{VC.Open}$ returns the standard Merkle path as witness, but the opening includes both value v_i and randomness r_i. The witness w contains the sibling hashes along the path from leaf to root (Alg. 1, ln. 18–Alg. 1, ln. 20). In the example of Fig. 1, opening index 3 provides, besides v_3 and r_3, the green-shaded siblings ($h_{\mathtt{011}}$, $h_{\mathtt{00}*}$, and $h_{\mathtt{1}**}$) of the blue-shaded path from leaf to root ($h_{\mathtt{010}}$, $h_{\mathtt{01}*}$, and $h_{\mathtt{0}**}$).

For verification, $\mathsf{VC.Verify}$ receives the claimed opening (v, r) as well as the Merkle path as witness w, and recomputes the root. It reconstructs the leaf hash using the opening: $h = \mathsf{H}(\mathrm{leaf}(i) \| v \| r)$ (Alg. 1, ln. 23). Then follows standard Merkle tree traversal (Alg. 1, ln. 24–Alg. 1, ln. 31), recomputing the hashes from the leaf to the root using the opening and the Merkle path. In the example of Fig. 1, verification of index 3 starts by computing $h_{\mathtt{010}}$ from the claimed (v_3, r_3), then uses the green-shaded witness to reconstruct the blue-shaded path through $h_{\mathtt{01}*}$ and $h_{\mathtt{0}**}$ to obtain h_{***} and finally verify against C.

Lemma 1. *The construction of Alg. 1 satisfies Def. 2.*

Proof of Lem. 1.

- **Correctness.** By construction, for honestly generated commitments, openings, and witnesses, the verification algorithm recomputes the identical hash chain from leaf to root. Specifically, $\mathsf{VC.Open}$ extracts the authentication path traversed during commitment, and $\mathsf{VC.Verify}$ reconstructs this path deterministically, yielding the same root hash C.
- **Binding.** Assume for contradiction that a PPT adversary $\mathcal{A}$ outputs $(\boldsymbol{v}, \boldsymbol{v}', \boldsymbol{r}, \boldsymbol{r}')$ with $\boldsymbol{v} \neq \boldsymbol{v}'$ or $\boldsymbol{r} \neq \boldsymbol{r}'$, but $\mathsf{VC.Commit}(\boldsymbol{v}, \boldsymbol{r}) = \mathsf{VC.Commit}(\boldsymbol{v}', \boldsymbol{r}') = C$. First, observe that both Merkle trees for $(\boldsymbol{v}, \boldsymbol{r})$ and $(\boldsymbol{v}', \boldsymbol{r}')$ have the same root hash C. Second, since $\boldsymbol{v} \neq \boldsymbol{v}'$ or $\boldsymbol{r} \neq \boldsymbol{r}'$, there must exist $i^* \in [L]$ where $v_{i^*} \neq v'_{i^*}$ or $r_{i^*} \neq r'_{i^*}$, so the leaf hashes at position i^* are computed from different inputs, $h_{\mathrm{leaf}(i^*)} = \mathsf{H}(\mathrm{leaf}(i^*) \| v_{i^*} \| r_{i^*})$ and $h'_{\mathrm{leaf}(i^*)} = \mathsf{H}(\mathrm{leaf}(i^*) \| v'_{i^*} \| r'_{i^*})$. Third, consider the paths from these leaves to the root in both trees. Since both paths reach the same root C, but start from different openings, there must exist some hash node b along these paths where the outputs of H are the same between the two Merkle trees, $h_b = h'_b$, even though the inputs differ. Namely, either b is the leaf node itself, or b is an interior node with $\mathsf{H}(b \| h_{b_\mathrm{L}} \| h_{b_\mathrm{R}}) = h_b = h'_b = \mathsf{H}(b \| h'_{b_\mathrm{L}} \| h'_{b_\mathrm{R}})$ even though $b \| h_{b_\mathrm{L}} \| h_{b_\mathrm{R}} \neq b \| h'_{b_\mathrm{L}} \| h'_{b_\mathrm{R}}$, for b_L and b_R the children of b. Fourth, in the random oracle model, finding such distinct inputs to H that produce the same output occurs with probability at most $q^2/2^\kappa$ (cf. birthday paradox) for an

Algorithm 1 Hiding Merkle tree (see Fig. 1) vector commitment scheme (see Def. 2), for simplicity described for vectors of length $L = 2^d$ for some fixed $d \in \mathbb{N}$, using random oracle H

Helper functions (see Fig. 1 for identification of hashes in the Merkle tree):

- leaf(i): Returns the d-bit binary representation of $i-1$ for $i \in [L]$ as an identifier for the corresponding leaf hash in the Merkle tree. E.g., for $d = 3$, leaf(3) = 010.
- ancestors(b): For leaf hash identifier $b \in \{0,1\}^d$, returns the identifiers of all hashes on the path from the leaf to the root of the Merkle tree, including the leaf but excluding the root. E.g., for $d = 3$, ancestors(010) = [010, 01$*$, 0$**$].
- parent(b): For the identifier b of a hash in the Merkle tree, returns the identifier of the parent node in the Merkle tree, by replacing the rightmost non-$*$ bit in b with $*$. E.g., for $d = 3$, parent(01$*$) = 0$**$.
- sibling(b): For the identifier b of a hash in the Merkle tree, returns the identifier of the sibling in the Merkle tree, by flipping the rightmost non-$*$ bit. E.g., for $d = 3$, sibling(01$*$) = 00$*$.

procedure MerkleTree($\boldsymbol{v} = (v_1, \ldots, v_L), \boldsymbol{r} = (r_1, \ldots, r_L)$)
  $\mathcal{T} \leftarrow \emptyset$
  for $i \in [L]$ // Compute leaf hashes
    $\mathcal{T}.h_{\mathrm{leaf}(i)} \leftarrow \mathsf{H}(\mathrm{leaf}(i) \| v_i \| r_i)$
  for $\ell \in [d-1, ..., 0]$ // Compute interior hashes
    for each interior node at level ℓ with hash identifier b (see Fig. 1)
      $b_{\mathrm{L}} \leftarrow$ left child of b
      $b_{\mathrm{R}} \leftarrow$ right child of b
      $\mathcal{T}.h_b \leftarrow \mathsf{H}(b \| \mathcal{T}.h_{b_{\mathrm{L}}} \| \mathcal{T}.h_{b_{\mathrm{R}}})$
  return $\mathcal{T}$

procedure VC.Commit($\boldsymbol{v} = (v_1, \ldots, v_L), \boldsymbol{r} = (r_1, \ldots, r_L)$)
  $\mathcal{T} \leftarrow$ MerkleTree($\boldsymbol{v}, \boldsymbol{r}$)
  return $C \leftarrow \mathcal{T}.h_{*\cdots*}$ // Merkle root

procedure VC.Open($\boldsymbol{v}, \boldsymbol{r}, i$)
  $\mathcal{T} \leftarrow$ MerkleTree($\boldsymbol{v}, \boldsymbol{r}$)
  $w \leftarrow \emptyset$
  for $b \in$ ancestors(leaf(i)) // Merkle path as witness
    $b' \leftarrow$ sibling(b)
    $w.h_{b'} \leftarrow \mathcal{T}.h_{b'}$
  return w

procedure VC.Verify(C, i, v, r, w)
  $h \leftarrow \mathsf{H}(\mathrm{leaf}(i) \| v \| r)$ // Re-compute leaf hash from opening
  for $b \in$ ancestors(leaf(i)) // Re-compute hashes from leaf to root using Merkle path
    $b_0 \leftarrow$ parent(b)
    $b' \leftarrow$ sibling(b)
    if rightmost non-$*$ bit in b is 0
      $h \leftarrow \mathsf{H}(b_0 \| h \| w.h_{b'})$
    else
      $h \leftarrow \mathsf{H}(b_0 \| w.h_{b'} \| h)$
  if $h = C$ // Check Merkle root
    return 1
  return 0

adversary making q queries to H. This probability is negligible in κ since $\mathcal{A}$ is PPT and thus q is polynomially bounded in κ.

- **Position binding.** The argument is similar to the binding proof. Suppose a PPT adversary $\mathcal{A}$ outputs $(C, i, v, v', r, r', w, w')$ with $v \neq v'$ or $r \neq r'$, where both $\mathsf{VC.Verify}(C, i, v, r, w) = 1$ and $\mathsf{VC.Verify}(C, i, v', r', w') = 1$. First, both verification paths successfully reach the same root C from position i. Second, since $v \neq v'$ or $r \neq r'$, the verification algorithm computes leaf hashes $h = \mathsf{H}(\text{leaf}(i)\|v\|r)$ and $h' = \mathsf{H}(\text{leaf}(i)\|v'\|r')$ from different inputs since $v \neq v'$ or $r \neq r'$. Third, consider the paths from these leaves to the roots in the Merkle trees using witnesses w and w'. Since both paths reach the same root C but start from different inputs, there must exist some node b along these paths where the outputs of H are equal, $h_b = h'_b$, even though the inputs differ (this holds even if $w = w'$). Namely, either b is the leaf node itself with different inputs from the openings producing the same hash, or b is an interior node where the hash computation yields the same output from different child hashes (even if $w = w'$). Fourth, in the random oracle model, finding such distinct inputs to H that produce the same output occurs with probability at most $q^2/2^\kappa$ (cf. birthday paradox) for an adversary making q queries to H. This probability is negligible in κ since $\mathcal{A}$ is PPT.
- **Hiding.** Consider the hiding game where adversary $\mathcal{A}$ provides $(\boldsymbol{v}^{(0)}, \boldsymbol{v}^{(1)})$. The challenger samples $b \xleftarrow{\$} \{0,1\}$ and $\boldsymbol{r} \xleftarrow{\$} \mathcal{R}^L$ uniformly at random, then gives $C \leftarrow \mathsf{VC.Commit}(\boldsymbol{v}^{(b)}, \boldsymbol{r})$ to $\mathcal{A}$. The adversary may adaptively query openings at indices i_j where $v_{i_j}^{(0)} = v_{i_j}^{(1)}$ and then receives $(v_{i_j}^{(b)}, r_{i_j}, w_j)$ with $w_j \leftarrow \mathsf{VC.Open}(\boldsymbol{v}^{(b)}, \boldsymbol{r}, i_j)$. To simplify the analysis, we assume that the adversary learns the randomness r_i for all indices i where $v_i^{(0)} = v_i^{(1)}$ as well as all the hashes in the Merkle tree. We show that the adversary's views for $b = 0$ and $b = 1$ are indistinguishable, and thus the adversary has no information about b and can only make a random guess $\hat{b}$.
  Since the challenger chooses $\boldsymbol{r}$ uniformly from $\mathcal{R}^L$ where $|\mathcal{R}| \geq 2^\kappa$, and the adversary is PPT (making at most polynomially many queries q to H), the probability that the adversary queries H on an input of the form $\text{leaf}(i)\|v_i^{(b)}\|r_i$ for any position i where $v_i^{(0)} \neq v_i^{(1)}$ is at most $q/2^\kappa$, which is negligible. Conditioning on this event not occurring, the adversary never queries H on the actual inputs used to compute leaf hashes at indices where the vectors differ.
  Under this conditioning, the distributions of all Merkle tree hashes are identical for $b = 0$ and $b = 1$. For indices i where $v_i^{(0)} = v_i^{(1)}$, the leaf hash $h_{\text{leaf}(i)}$ is identical for both values of b since both the value and randomness are the same. For indices i where $v_i^{(0)} \neq v_i^{(1)}$, since the adversary has not queried H on the corresponding inputs $\text{leaf}(i)\|v_i^{(0)}\|r_i$ and $\text{leaf}(i)\|v_i^{(1)}\|r_i$, the random oracle returns uniform and independent hashes $h_{\text{leaf}(i)} \xleftarrow{\$} \{0,1\}^\kappa$, irrespective of $v_i^{(b)}$. Since the leaf hashes are identically distributed for both values of b, and the Merkle tree construction deterministically combines leaf hashes to produce hashes at interior nodes, all hashes in the tree (including the root) are identically distributed irrespective of b.
  Thus, in the overwhelmingly probable case where the adversary does not query H on the actual inputs used to compute leaf hashes at indices where the vectors differ, even if the adversary knows all hashes of the entire Merkle tree of $\boldsymbol{v}^{(b)}$ with r, and the randomness r_i for all indices i where $v_i^{(0)} = v_i^{(1)}$, the missing randomness at indices where the vectors differ prevents the adversary from distinguishing which vector was committed by the challenger.

□

Algorithm 2 Hiding erasure-correcting code (HECC) instantiation (see Defs. 3 and 4)

// Parameters: integers K, T, N with $N \geq K + T$, finite field $\mathbb{F}$ with $|\mathbb{F}| \geq N$.
// Fix distinct field elements $\alpha_1, ..., \alpha_N \in \mathbb{F}$ as *evaluation points*.

procedure $\mathsf{HECC.Enc}(\boldsymbol{m} = (m_1, ..., m_K) \in \mathbb{F}^K, \boldsymbol{r} = (r_1, ..., r_T) \in \mathbb{F}^T)$
  $f(X) \leftarrow \sum_{i=1}^{K} m_i X^{i-1} + \sum_{j=1}^{T} r_j X^{K-1+j}$ // Define polynomial $f(X)$
  for $i = 1, ..., N$
    $c_i \leftarrow f(\alpha_i)$ // Evaluate polynomial $f(X)$ at α_i
  return $\boldsymbol{c} \leftarrow (c_1, ..., c_N)$

procedure $\mathsf{HECC.Dec}((c_{i_1}, i_1), ..., (c_{i_{K+T}}, i_{K+T}))$
  assert $\forall j \neq k : i_j \neq i_k$ // Decoding requires indices to be distinct
  $\hat{f}(X) \leftarrow \sum_{j=1}^{K+T} c_{i_j} \prod_{\substack{k=1 \\ k \neq j}}^{K+T} \frac{X - \alpha_{i_k}}{\alpha_{i_j} - \alpha_{i_k}}$ // Compute polynomial $\hat{f}(X)$ of degree at most $K + T - 1$ using Lagrange interpolation of $((\alpha_{i_1}, c_{i_1}), ..., (\alpha_{i_{K+T}}, c_{i_{K+T}}))$
  for $i = 1, ..., K$
    $\hat{m}_i \leftarrow$ coefficient of X^{i-1} in $\hat{f}(X)$ // Extract message coefficients
  for $j = 1, ..., T$
    $\hat{r}_j \leftarrow$ coefficient of X^{K-1+j} in $\hat{f}(X)$ // Extract randomness coefficients
  return $(\hat{\boldsymbol{m}}, \hat{\boldsymbol{r}}) \leftarrow ((\hat{m}_1, ..., \hat{m}_K), (\hat{r}_1, ..., \hat{r}_T))$

2.1.3 Hiding Erasure-Correcting Codes

Leveraging the similarity between Reed–Solomon erasure-correcting codes [62] and secret sharing [65, 12, 29, 41], allows us to implement the following *hiding erasure-correcting code* (HECC) primitive, where a *message* is encoded into *shreds* in such a way that (i) "few" shreds do not reveal any information about the encoded message, while (ii) from "enough"/"many" shreds, the message can reliably be reconstructed:

Definition 3 (HECC Syntax)**.** For non-negative integers N, K, T with $N \geq K + T$, and a finite field $\mathbb{F}$ with $|\mathbb{F}| \geq N$, a hiding erasure-correcting code HECC provides two algorithms:

- **Encoding** $\mathsf{HECC.Enc}$: $(\boldsymbol{m} \in \mathbb{F}^K, \boldsymbol{r} \in \mathbb{F}^T) \to (\boldsymbol{c} = (c_1, ..., c_N) \in \mathbb{F}^N)$**.** Given a *message* $\boldsymbol{m}$, represented as a vector of K field elements, and *randomness* $\boldsymbol{r}$, represented as a vector of T field elements, as inputs, $\mathsf{HECC.Enc}$ deterministically outputs a vector $\boldsymbol{c}$ of N field elements $c_1, ..., c_N \in \mathbb{F}$, called *shreds*.
- **Decoding** $\mathsf{HECC.Dec}$: $((c_{i_j}, i_j)_{j=1}^{K+T} \in (\mathbb{F} \times \mathbb{N})^{K+T}) \to (\hat{\boldsymbol{m}} \in \mathbb{F}^K, \hat{\boldsymbol{r}} \in \mathbb{F}^T)$**.** Given a set of $K+T$ pairs of shreds c_{i_j} and their corresponding indices i_j as inputs, $\mathsf{HECC.Dec}$ outputs a message $\hat{\boldsymbol{m}}$ and randomness $\hat{\boldsymbol{r}}$.

Definition 4 (HECC Properties)**.** A proper HECC satisfies the following two properties:

- **Erasure-correction.** For every message $\boldsymbol{m} \in \mathbb{F}^K$ and every randomness $\boldsymbol{r} \in \mathbb{F}^T$, let $\boldsymbol{c} = (c_1, ..., c_N) \leftarrow \mathsf{HECC.Enc}(\boldsymbol{m}, \boldsymbol{r})$. Then, for any set of $K + T$ pairs (c_{i_j}, i_j) where the indices are distinct, $\mathsf{HECC.Dec}((c_{i_j}, i_j)_{j=1}^{K+T}) = (\boldsymbol{m}, \boldsymbol{r})$.
- **Hiding.** For every two messages $\boldsymbol{m}, \boldsymbol{m'} \in \mathbb{F}^K$, let $\boldsymbol{r}, \boldsymbol{r'} \xleftarrow{\$} \mathbb{F}^T$, and let $\boldsymbol{c} = (c_1, ..., c_N) \leftarrow \mathsf{HECC.Enc}(\boldsymbol{m}, \boldsymbol{r})$, and $\boldsymbol{c'} = (c'_1, ..., c'_N) \leftarrow \mathsf{HECC.Enc}(\boldsymbol{m'}, \boldsymbol{r'})$. Then, for any set of T distinct indices i_j, $(c_{i_1}, ..., c_{i_T})$ and $(c'_{i_1}, ..., c'_{i_T})$ are identically distributed. This implies that an adversary with at most T shreds cannot learn any information about the underlying message.

  Specifically, we can construct an HECC from a traditional maximum-distance separable (MDS)

$(K+T, N)$ erasure-correcting code such as Reed–Solomon, by sampling T random field elements and appending them to the original message $\boldsymbol{m} \in \mathbb{F}^K$ before applying the code's encoding. This construction is carried out explicitly, with Reed–Solomon codes, in Alg. 2.

Lemma 2. *Alg. 2 provides a HECC as per Defs. 3 and 4.*

Proof of Lem. 2.
- **Erasure-correction.** The polynomial $f(X)$ of Alg. 2, ln. 4 has degree at most $K+T-1$, so it is uniquely determined by evaluations at any $K+T$ distinct points, and thus for $\hat{f}(X)$ of Alg. 2, ln. 10, $f = \hat{f}$. Therefore, both $\hat{\boldsymbol{m}} = \boldsymbol{m}$ (from coefficients of $X^0, ..., X^{K-1}$ in $\hat{f}$) and $\hat{\boldsymbol{r}} = \boldsymbol{r}$ (from coefficients of $X^K, ..., X^{K+T-1}$ in $\hat{f}$).
- **Hiding.** We show that for any fixed message $\boldsymbol{m}$, any T evaluations (at distinct points) of the polynomial $f(X)$ of Alg. 2, ln. 4 are distributed independently and uniformly random over $\mathbb{F}$, if $\boldsymbol{r}$ is sampled randomly. This implies that shreds, at any T distinct indices, of two messages $\boldsymbol{m}$ and $\boldsymbol{m}'$ are identically distributed for independently random $\boldsymbol{r}, \boldsymbol{r}'$, and thus establishes hiding. So consider any fixed message $\boldsymbol{m} = (m_1, ..., m_K)$ and set of T distinct indices i_j. Then,

$$\forall j \in [T]: \quad c_{i_j} = f(\alpha_{i_j}) = \sum_{k=1}^{K} m_k \alpha_{i_j}^{k-1} + \sum_{j=1}^{T} r_j \alpha_{i_j}^{K-1+j}.$$

  Thus, we can write the shreds at the distinct indices i_j as a sum of two matrix-vector products:

$$\begin{bmatrix} c_{i_1} \\ \vdots \\ c_{i_T} \end{bmatrix} = \begin{bmatrix} \alpha_{i_1}^0 & \dots & \alpha_{i_1}^{K-1} \\ \vdots & \ddots & \vdots \\ \alpha_{i_T}^0 & \dots & \alpha_{i_T}^{K-1} \end{bmatrix} \begin{bmatrix} m_1 \\ \vdots \\ m_K \end{bmatrix} + \underbrace{\begin{bmatrix} \alpha_{i_1}^K & \dots & \alpha_{i_1}^{K+T-1} \\ \vdots & \ddots & \vdots \\ \alpha_{i_T}^K & \dots & \alpha_{i_T}^{K+T-1} \end{bmatrix}}_{\triangleq M_{\text{masking}}} \begin{bmatrix} r_1 \\ \vdots \\ r_T \end{bmatrix}.$$

  Since M_{masking} is a $T \times T$ Vandermonde matrix, it is full-rank over $\mathbb{F}$. Furthermore, $r_1, ..., r_T$ are sampled independently and uniformly random over $\mathbb{F}$. As a result, the term $M_{\text{masking}}\boldsymbol{r}$ is uniformly random over $\mathbb{F}^T$, and the shreds $(c_{i_1}, ..., c_{i_T})$ are distributed independently and uniformly random over $\mathbb{F}$, *independent of the message* $\boldsymbol{m}$. Thus, at most T shreds provide no information about the message $\boldsymbol{m}$.

□

It is clear that the HECC of Alg. 2 can trivially be "vectorized"/"batched" (see [57, Fig. 5]), in the spirit of interleaved Reed–Solomon codes [1], to accommodate long messages.

2.1.4 Combining Vector Commitments and Hiding Erasure-Correcting Codes

Our protocol requires vector commitments with $\mathcal{V} = \mathcal{R} = \mathbb{F}$ and $L = N$ that, even if the masking randomness is chosen as a length-N HECC encoding of a uniformly random length-K message with uniformly random length-T randomness (both in $\mathbb{F}$, and with $K, T > 0$), remains hiding to any PPT adversary that knows at most T openings of the vector commitment. We show that this is indeed the case for Algs. 1 and 2.

Lemma 3. *Consider Alg. 2. Let* $\boldsymbol{m} \xleftarrow{\$} \mathbb{F}^K$, $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^T$, $\boldsymbol{c} = (c_1, ..., c_N) \leftarrow \mathsf{HECC.Enc}(\boldsymbol{m}, \boldsymbol{r})$. *Then, for any set of* $K+T$ *distinct indices* i_j, $(c_{i_1}, ..., c_{i_{K+T}})$ *is uniformly random over* $\mathbb{F}^{K+T}$.

Proof of Lem. 3. The proof is analogous to the proof of the hiding property for Lem. 2. For any set of $K+T$ distinct indices i_j, we can write the shreds encoded according to Alg. 2 as:

$$\begin{bmatrix} c_{i_1} \\ \vdots \\ c_{i_{K+T}} \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha_{i_1}^0 & \cdots & \alpha_{i_1}^{K+T-1} \\ \vdots & \ddots & \vdots \\ \alpha_{i_{K+T}}^0 & \cdots & \alpha_{i_{K+T}}^{K+T-1} \end{bmatrix}}_{\triangleq M_{\text{encoding}}} \begin{bmatrix} m_1 \\ \vdots \\ m_K \\ r_1 \\ \vdots \\ r_T \end{bmatrix}$$

The matrix M_{encoding} is a $(K+T)\times(K+T)$ Vandermonde matrix with distinct α_{i_j}, hence invertible over $\mathbb{F}$. Since $\boldsymbol{m} \xleftarrow{\$} \mathbb{F}^K$ and $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^T$ are sampled independently and uniformly at random, the concatenated vector $(m_1, ..., m_K, r_1, ..., r_T)$ is uniformly random over $\mathbb{F}^{K+T}$. Because M_{encoding} is an invertible linear transformation and the input vector is uniformly random over $\mathbb{F}^{K+T}$, the output vector $(c_{i_1}, ..., c_{i_{K+T}})$ is also uniformly random over $\mathbb{F}^{K+T}$. □

Lemma 4. *Consider Alg. 2 with $K, T > 0$, and Alg. 1 with $L = N$, $\mathcal{V} = \mathcal{R} = \mathbb{F}$. Every PPT adversary $\mathcal{A}$ succeeds in the following game with probability negligible in κ: The adversary $\mathcal{A}$ provides $(\boldsymbol{v}^{(0)}, \boldsymbol{v}^{(1)})$. The challenger flips an unbiased coin $b \xleftarrow{\$} \{0,1\}$, samples $\boldsymbol{m'} \xleftarrow{\$} \mathbb{F}^K$ and $\boldsymbol{r'} \xleftarrow{\$} \mathbb{F}^T$, computes $\boldsymbol{r} = (r_1, ..., r_N) \leftarrow \mathsf{HECC.Enc}(\boldsymbol{m'}, \boldsymbol{r'})$, and gives $C \leftarrow \mathsf{VC.Commit}(\boldsymbol{v}^{(b)}, \boldsymbol{r})$ to $\mathcal{A}$. The adversary $\mathcal{A}$ may make up to T queries i_j for distinct indices where $v_{i_j}^{(0)} = v_{i_j}^{(1)}$ and receives openings $(r^{(j)}, w^{(j)})$ where $r^{(j)} \leftarrow r_{i_j}$ and $w^{(j)} \leftarrow \mathsf{VC.Open}(\boldsymbol{v}^{(b)}, \boldsymbol{r}, i_j)$. Finally, $\mathcal{A}$ outputs a guess $\hat{b} \in \{0,1\}$. The adversary succeeds iff $\hat{b} = b$.*

Proof of Lem. 4. The proof follows a similar approach as the proof of the hiding property for Lem. 1. Consider the hiding game where adversary $\mathcal{A}$ provides $(\boldsymbol{v}^{(0)}, \boldsymbol{v}^{(1)})$. The challenger samples $b \xleftarrow{\$} \{0,1\}$, $\boldsymbol{m'} \xleftarrow{\$} \mathbb{F}^K$ and $\boldsymbol{r'} \xleftarrow{\$} \mathbb{F}^T$, computes $\boldsymbol{r} = (r_1, ..., r_N) \leftarrow \mathsf{HECC.Enc}(\boldsymbol{m'}, \boldsymbol{r'})$, and gives $C \leftarrow \mathsf{VC.Commit}(\boldsymbol{v}^{(b)}, \boldsymbol{r})$ to $\mathcal{A}$. The adversary may query openings at up to T distinct indices i_j where $v_{i_j}^{(0)} = v_{i_j}^{(1)}$ and receives (r_{i_j}, w_j) with $w_j \leftarrow \mathsf{VC.Open}(\boldsymbol{v}^{(b)}, \boldsymbol{r}, i_j)$.

Let Q denote the set of indices queried by the adversary for openings, with $|Q| \leq T$. So the adversary learns (r_i, w_i) with $w_i \leftarrow \mathsf{VC.Open}(\boldsymbol{v}^{(b)}, \boldsymbol{r}, i)$ for all $i \in Q$. Without loss of generality, assume that the adversary learns all hashes of the Merkle tree computed as part of $\mathsf{VC.Commit}(\boldsymbol{v}^{(b)}, \boldsymbol{r})$. We show that the adversary's views for $b = 0$ and $b = 1$ are indistinguishable.

First, observe that the adversary learns $|Q|$ shreds $\{r_i \mid i \in Q\}$ from the HECC encoding $\boldsymbol{r} = (r_1, ..., r_N) \leftarrow \mathsf{HECC.Enc}(\boldsymbol{m'}, \boldsymbol{r'})$. By Lem. 3, since $\boldsymbol{m'}$ and $\boldsymbol{r'}$ are uniformly random, any $K+T$ shreds are jointly uniform over $\mathbb{F}^{K+T}$. Since $K > 0$ and $|Q| \leq T$, this implies that any $|Q|+1$ shreds are jointly uniform. Therefore, for any index $i \notin Q$, the value r_i is uniformly distributed over $\mathbb{F}$ even conditioned on the observed shreds $\{r_i \mid i \in Q\}$.

Now consider any index i where $v_i^{(0)} \neq v_i^{(1)}$. Since the adversary can only query indices for opening where $v_i^{(0)} = v_i^{(1)}$, we have $i \notin Q$. By the above, r_i is uniform over $\mathbb{F}$ from the adversary's perspective *a priori*. The leaf hash $h_{\text{leaf}(i)}$ at index i is computed as $\mathsf{H}(\text{leaf}(i) \| v_i^{(b)} \| r_i)$. Since the adversary is PPT, it can make at most $q = \text{poly}(\kappa)$ queries to the random oracle H. Since

$|\mathbb{F}| \geq 2^\kappa$, the probability of the event (denote the event by E) that the adversary ever queries H on $\text{leaf}(i)\|v_i^{(b)}\|r_i$ for any index i where $v_i^{(0)} \neq v_i^{(1)}$ is at most $q/|\mathbb{F}| \leq q/2^\kappa = \text{negl}(\kappa)$.

Conditioned on $\neg E$, the distribution of all Merkle tree hashes is identical for $b = 0$ and $b = 1$: At indices i where $v_i^{(0)} = v_i^{(1)}$, the leaf hash $h_{\text{leaf}(i)}$ is identical for both values of b (same value and randomness). At indices i where $v_i^{(0)} \neq v_i^{(1)}$, the adversary cannot learn r_i through opening, and since the adversary cannot have queried $\mathsf{H}(\text{leaf}(i)\|v_i^{(b)}\|r_i)$, the leaf hash $h_{\text{leaf}(i)}$ is uniform and independent, regardless of b. Since interior hashes are computed deterministically from leaf hashes, all Merkle tree hashes including C have identical distribution for both values of b. □

2.2 Setting, Adversary, Network

For ease of exposition, we consider a standard *permissioned* setting [51] with a computationally bounded adversary and a *partially synchronous* network [37], which we recall below.

Setting We consider a setting with n *nodes*, denoted by a set $\mathcal{N}$. Each node has a cryptographic identity that is known to all nodes. *Time* in the considered environment proceeds continuously, indexed by t, and nodes are assumed to have synchronized clocks. In contrast to this notion of "real time" provided by the environment, the protocols we consider proceed in "logical steps" called *slots*, indexed by s. The environment has two more features which we detail below: a PPT *adversary* that attempts to subvert the proper functioning of the protocol, and a *network* functionality, with which nodes can send each other messages as instructed by the protocol.

Adversary Part of every execution is a PPT adversary that attempts to disturb the proper functioning of the protocol (specifically, it seeks to prevent the protocol from guaranteeing the consensus security properties outlined in Sec. 2.3). For this purpose, for ease of exposition, we assume that the adversary chooses a set $\mathcal{N}_\text{a} \subseteq \mathcal{N}$ of $f \triangleq |\mathcal{N}_\text{a}|$ nodes to *corrupt* at the beginning of the execution before any of the protocol's randomness is drawn (*static corruption*). The adversary can cause these so-called *Byzantine* nodes to deviate arbitrarily from the protocol for the entire execution. Recall that in all its actions, the adversary is computationally bounded to be probabilistic polynomial-time (PPT). Non-adversary *honest* nodes, denoted by $\mathcal{N}_\text{h} \triangleq \mathcal{N} \setminus \mathcal{N}_\text{a}$, follow the protocol as specified. The adversary also determines the delays incurred by messages sent among honest nodes, as detailed next.

Network Nodes can send messages to each other via a fully-connected network of private point-to-point links. The network is *partially synchronous* [37], meaning there is a delay upper-bound Δ, known to all nodes, and a "global stabilization time" GST, chosen adaptively by the adversary during execution, such that before GST, the adversary can cause arbitrary message delays, while after GST, the message delay is adversarial but at most Δ. In particular, this means that every message sent by an honest node by t is delivered to all honest nodes by $\max(t, \mathsf{GST}) + \Delta$.

2.3 Consensus Basics

2.3.1 Safety and Liveness of Atomic Broadcast

We are interested in constructing protocols that solve the *atomic broadcast* variant of *consensus*, where, continuously over time, nodes are input *payloads* by the environment, and they output sequences of payloads considered *confirmed*, in so-called *logs*. Intuitively, the consensus protocol should ensure that (i) logs output by honest nodes are *consistent* across nodes and across time (*safety*), and (ii) every input payload eventually makes it into the log of every honest node (*liveness*). More formally, if we denote by L_t^p the log output by node p at time t, and if $L \preceq L'$ denotes that the sequence L is a prefix of (or equal to) the sequence L', then we require:

Definition 5 (Safety)**.** For all honest nodes p, p' and times t, t', we have $L_t^p \asymp L_{t'}^{p'}$, where $L_t^p \asymp L_{t'}^{p'}$ denotes that either $L_t^p \preceq L_{t'}^{p'}$ or $L_{t'}^{p'} \preceq L_t^p$.

Definition 6 (Liveness)**.** For every payload tx input to an honest node, there exists a time t_0, such that for all times $t \geq t_0$, for every honest node p, we have $\mathsf{tx} \in L_t^p$.

Definition 7 (Security)**.** An atomic broadcast protocol is *secure* with *resilience* τ iff in every partially synchronous execution with $f/n \leq \tau$, except with probability negligible in some cryptographic security parameter κ, the protocol is safe and live.

Note that safety implies that the logs output by honest nodes are all prefixes of one "ground truth" log. It is therefore meaningful to speak of *the log* output by a protocol. Specifically, for a safe protocol producing output logs L_t^p, we denote by L_∞^* "the log of the protocol", where L_∞^* is the shortest log with the property that $\forall p, t : L_t^p \preceq L_\infty^*$.

2.3.2 Slightly Enriched Interface

Our protocol will be constructed using, as an opaque-box, an off-the-shelf partially synchronous consensus protocol Π_{ab}. For both our protocol and for Π_{ab}, we assume an interface that is slightly enriched over the vanilla atomic broadcast primitive recalled above, namely in the following way. We assume that the protocol internally proceeds in "logical steps" called *slots* (terminology borrowed from Alpenglow [49]). Each slot has an associated *leader* node $\mathcal{L}(s)$ that gets to propose a payload for inclusion in the log, and each slot has an associated fixed *slot proposal time* T_s, where the leader is expected to produce and broadcast its proposal. Payloads confirmed in the log are annotated with a slot number (the payload is said to be *confirmed at that slot*) (similar to [71, Def. 9]). If an honest leader proposes a payload that is subsequently confirmed, this payload will be annotated with the slot in which it was proposed. In any case, honest nodes agree on the slot numbers of confirmed payloads, and the slot numbers are non-decreasing along the log.

Slightly more formally, we consider consensus protocols Π with the following interface:

- Protocol Π is parameterized by a leader sequence $\mathcal{L}(s)$.
- Protocol Π comes with a fixed slot proposal time sequence T_s.
- By time T_s, leader $\mathcal{L}(s)$ of slot s invokes $\Pi.\mathsf{Input}(s, \mathsf{txs})$ to propose payload txs in slot s.
- As payloads get confirmed, the protocol emits events $\Pi.\mathsf{Output}(s, \mathsf{txs})$, consecutively for $s = 1, 2, \ldots$. Here, txs is the payload confirmed at slot s, or $\perp$ (if no payload is confirmed at slot s, for instance because $\mathcal{L}(s)$ did not invoke $\Pi.\mathsf{Input}(s, \mathsf{txs})$ in time, or before GST).

It is important to highlight that the above is primarily a *syntactic* variation of consensus that will subsequently be convenient, rather than a change to the consensus security *semantics*. (Virtually) all partially synchronous candidate protocols for Π_{ab} we are aware of are easily modified to satisfy this refined interface. They already have a corresponding notion of slots: PBFT [25], Tendermint [20], or HotStuff [80] calls them *views*, Simplex [26] calls them *heights*, Streamlet [27] calls them *epochs*. They are easily modified to have a fixed regular cadence of slot proposal times, for instance every 2Δ time, and to annotate confirmed payloads with the slot number in which they were proposed (over which these protocols also establish consensus).

For any log L annotated with slot numbers, we denote by $L[s]$ the payloads in L confirmed *at slot* s, and we denote by $L[:s]$ the prefix of L of payloads confirmed *at any slot* $s' \leq s$.

Most partially synchronous candidate protocols for Π_{ab}, like PBFT, Tendermint, (non-chained) HotStuff, or Simplex, readily satisfy the following liveness guarantee:

Definition 8 (Leader-Driven Liveness)**.** There exists $c \geq 0$ such that for every slot s with $T_s \geq \mathsf{GST} + c\Delta$, if $\mathcal{L}(s) \in \mathcal{N}_{\text{h}}$ and $\mathcal{L}(s)$ invokes $\Pi.\mathsf{Input}(s, \mathsf{txs})$ no later than T_s, then $\mathsf{txs} \subseteq L^*_\infty[:s]$.

2.4 Economic Consensus Properties

To achieve the goals outlined in Sec. 1, we require, in addition to traditional safety and liveness, two new *economically-motivated* consensus properties that we define below. First, we need a stronger liveness-like notion, called *selective-censorship resistance*, that guarantees that transactions are not just *eventually* confirmed, but are confirmed at the next possible opportunity. Second, we need a privacy-like notion, called *hiding*, that guarantees that the adversary cannot learn about pending transactions before it is "too late" for the adversary to act on the information in the transactions.

2.4.1 Selective-Censorship Resistance

Intuitively, the leader $\mathcal{L}(s)$ drives consensus during slot s. If $\mathcal{L}(s)$ does not follow the protocol—for instance, if $\mathcal{L}(s)$ crashes during the slot—it becomes hard to satisfy that transactions input by T_s are confirmed at slot s. We thus only require that *either* all transactions input to honest nodes by T_s are confirmed at slot s, *or no* transactions are confirmed at slot s at all.

Definition 9 (Selective-Censorship Resistance)**.** A safe consensus protocol with output logs L^p_t is *selective-censorship resistant* iff for every slot s with $T_s \geq \mathsf{GST}$, it is the case that either $L^*_\infty[s] = \emptyset$, or for every transaction tx input to some honest node by some time $t \leq T_s$, $\mathsf{tx} \in L^*_\infty[:s]$ (or both).

This means that in order for the adversary to censor *any* transactions in a given slot, the adversary has to censor *all* transactions for that slot. Def. 9 is similar to the notion of *short-term censorship resistance* defined in [3].

2.4.2 Hiding

In addition to selective-censorship resistance, we require that the adversary cannot learn information about pending transactions input at honest nodes, before the transactions' eventual inclusion (and positions) in the protocol's log is decided. To make this notion precise, we build on the well-known concept of *valency* [5, 39]. The valency of a consensus protocol at a particular

point in the protocol's execution captures consensus decisions that are inevitable (no matter the adversary's future actions), given the state of honest nodes and of the environment at that point in the execution. Importantly, the valency captures not only consensus decisions that honest nodes have recognized (and externalized in their output logs) yet, but also comprises consensus decisions that have been "effectively made" by the protocol but not yet detected by honest nodes.

Definition 10 (Valency)**.** The *valency* $L_t^{\times}$ of a consensus protocol at a particular point t in an execution is the longest annotated log L such that, for all adversarial strategies for the continuation of the execution, it holds that $L \preceq L_{\infty}^{*}$.

A consensus protocol is hiding if the adversary cannot learn any information about honest nodes' input transactions before the transactions enter the valency of the protocol.

Definition 11 (Hiding)**.** A consensus protocol is *hiding* iff every PPT adversary $\mathcal{A}$ has no more than a negligible advantage over a random guess in winning the following game: The adversary $\mathcal{A}$ selects an honest node p and two transactions $\mathsf{tx}_0, \mathsf{tx}_1$. The challenger flips an unbiased local coin $b \xleftarrow{\$} \{0,1\}$, and provides node p with input tx_b. The protocol's execution commences. Let t^* be the first time such that $\mathsf{tx}_b \in L_{t^*}^{\times}$, where $L_{t^*}^{\times}$ denotes the valency of the protocol at time t^*. At this point t^*, the adversary $\mathcal{A}$ is asked to produce $\hat{b}$, and $\mathcal{A}$ wins the game iff $\hat{b} = b$.

2.4.3 Practical Considerations of Selective-Censorship Resistance and Hiding

To discuss practical considerations, it is important to recognize how the abstraction used for the formal parts of this paper matches with the intended application setting. In a blockchain like Solana or Ethereum, users submit transactions to infrastructure providers such as RPCs or block builders. Our abstraction does not capture users or RPCs/block builders. Instead, it treats nodes as both the entities where transactions are input, and the entities that run the consensus protocol.

As a result, Def. 11 differs slightly from the hiding notions used for instance in encrypted mempools, in a practically relevant way: Encrypted mempools guarantee to users the "privacy" of their transactions from all nodes. The result is inefficient use of the scarce resource "block space", because nodes cannot filter spam or ensure that transactions are able to pay fees. In contrast, Def. 11 does not guarantee "privacy" to users, it only guarantees that nodes cannot learn the private transaction flow of other nodes before the corresponding transactions are confirmed. This allows RPCs/block builders to select high-value transactions and allocate block space efficiently.

In practice already today, users effectively enjoy "privacy" due to the trust relationships and repeated-game dynamics with their corresponding RPCs/block builders (this is not captured in our model). An advantage of a hiding protocol is that there are multiple nodes a user could route its transaction flow to. A user can ensure to only employ nodes that respect the user's privacy.

Note that a protocol being hiding without also being selective-censorship resistant does not help with the economic properties we desire. If a protocol is not selective-censorship resistant, an adversary can selectively censor all transactions that are not input to the adversary. Thereby, the adversary effectively coerces users to submit transactions through the adversary, so that de-facto all input transactions are revealed to the adversary (undermining the hiding property).

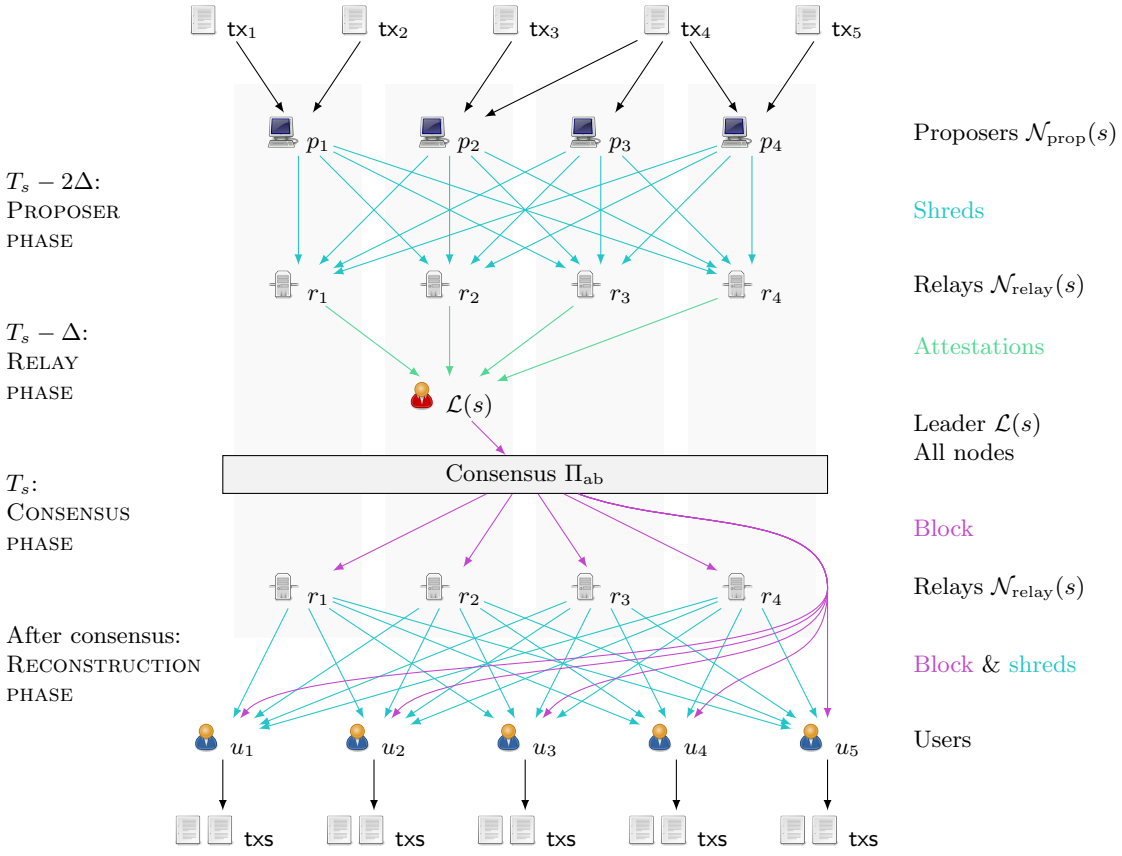

Figure 2: Overview of the Multiple Concurrent Proposers (MCP) protocol (Algs. 3 and 4 and Sec. 3), showing one slot s. The protocol operates in four main phases: (1) PROPOSER PHASE: Proposers $\mathcal{N}_{\text{prop}}(s)$ collect transactions into batches, encode the batches into shreds, and send the shreds to relays $\mathcal{N}_{\text{relay}}(s)$ privately. (2) RELAY PHASE: Relays $\mathcal{N}_{\text{relay}}(s)$ verify and store shreds, and send attestations (about shreds they have received) to the consensus leader $\mathcal{L}(s)$. (3) CONSENSUS PHASE: The leader $\mathcal{L}(s)$ aggregates relay attestations into a block, which is input to the consensus protocol Π_{ab} run by all nodes. (4) RECONSTRUCTION PHASE: After consensus, the relays broadcast stored shreds. All nodes decode available batches from the shreds, and output the transaction batches.

3 Multiple Concurrent Proposers Protocol

3.1 Philosophy & Overview

In traditional consensus protocols, a single *leader* per slot collects transactions into a block for proposal as the next consensus decision. This leaves the leader with outsized power over transaction inclusion, which is detrimental to the economic properties of the system. Our multiple concurrent proposers (MCP) protocol modifies this approach (see Fig. 2): instead of one leader proposing all the transactions for a slot, multiple *proposers* each submit a *batch* of transactions for the same slot. Then, a committee of *relays* mediates between proposers and the consensus leader.

Relays receive *shreds*, HECC-encoded pieces of batches, and create *attestations* that certify the timely receipt of shreds and vouch for the *availability* of the batches to the leader. As a result, the leader can incorporate batches by reference into its block, *even before* learning the batches' contents, and with confidence that once consensus is reached, these batches can be reconstructed from the shreds stored at relays. In addition, the relays constrain the leader's degrees of freedom in choosing which batches to include in the block, by requiring that the leader include attestations from "many" relays to form a valid block, which in turn forces the leader to incorporate the batches of honest proposers.

It is important to note that proposers, relays, and consensus nodes are separate roles and the corresponding nodes could be chosen from different sets. For ease of exposition, however, we describe here the scenario where each of the system-wide set of n are eligible for each role. We also assume that all of the n nodes act as proposers for every slot. In practice, the set of proposers would be subsampled from the set of all nodes. Obviously, then our selective-censorship resistance property applies only to transactions that are submitted to honest proposers.

In more detail, our MCP protocol operates in four distinct phases per slot. In the *proposer phase*, each proposer collects transactions into a batch, encodes it using HECC into multiple shreds, and distributes a distinct shred to each relay. During the *relay phase*, relays validate and store received shreds without learning the contents of the underlying batches, then create attestations that certify timely receipt and availability of batches' shreds to the consensus leader. The *consensus phase* sees the leader aggregate these attestations into a consensus block. Batches are implicitly confirmed if sufficiently many relay attestations attest to the batches' shreds being available. Then, the underlying consensus protocol runs, and, once the block is confirmed, the *reconstruction phase* begins. In this phase, relays broadcast their stored shreds so that nodes can reconstruct all the confirmed transaction batches. Nodes then take the union of transactions included in the batches and orders them by a deterministic rule (e.g., by priority fee) to determine the order they are added to those nodes' logs.

Selective-Censorship Resistance Since the consensus leader must include attestations from many relays to obtain a valid block, the leader is forced to include attestations from a significant number of honest relays. Attestations from honest relays will reference all honest proposals (under synchrony). We choose the threshold for the maximum amount of attestations the leader can exclude such that we are guaranteed sufficiently many honest relay attestations to be able to confirm all honest proposals (during synchrony). Thus the leader's only way to censor a target batch is to not propose any block at all.

Hiding Proposers encode batches using HECC with parameters that ensure that shreds stored at adversarial relays reveal no information about batch contents. Additionally, honest relays only broadcast their shreds after consensus has been reached on transaction batch inclusion. As a result, no node (other than the originating proposer) can observe or adapt to transaction contents before batches are confirmed.[5]

[5]Notably, our protocol does not solve the problem of obscuring transactions from the proposer. The concern regarding a proposer's potential adverse selection against a user after seeing their bid in an auction is mitigated by the repeated nature of the interaction between proposers and users. In a competitive environment with multiple proposers, users may switch proposer if one proposer repeatedly acts against their interests.

3.2 Detailed Description

We now describe the MCP protocol construction in detail. Pseudo-code is provided in Algs. 3 and 4. The protocol consists of a gadget Π_{mcp} that can be applied to any standard consensus protocol Π_{ab} (that satisfies the enriched interface described in Sec. 2.3) so that the composite $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ satisfies the economic properties described in Sec. 2.4. After setup, the protocol iterates through slots, each of which consists of four phases, which we detail below.

3.2.1 Setup (Alg. 3, ln. 1)

The protocol construction has three main parameters (Alg. 3, ln. 1): *coding rate* $\gamma \in [0, 1]$, *relay threshold* $\mu \in [0, 1]$, and *availability threshold* $\varphi \in [0, 1]$. The construction is also parameterized by the target resilience τ of $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$. The relay threshold μ determines the fraction of relays from which an attestation must be included to form a valid consensus block (Alg. 4, ln. 9). This ensures that adversarial consensus leaders cannot get away without including attestations from "many" honest relays (and, thereby, about "many" honest proposers' batches, when the network is synchronous). The availability threshold φ determines the fraction of relays that must have attested for a proposer's batch for said batch to be considered available and required to be included in the protocol's output log (Alg. 4, ln. 20). This ensures that only batches for which it is guaranteed that they can be reconstructed from the relays' shreds are considered confirmed. This threshold is higher than the coding rate to account for the fact that adversarial relays might attest to a batch but never release the corresponding shreds.

All nodes share cryptographic parameters established during setup (Alg. 3, ln. 2). Each node p has a key pair $(\mathsf{sk}_p, \mathsf{pk}_p)$ for digital signatures, where all public keys are common knowledge. For each slot s, the protocol randomly samples (Alg. 3, ln. 3) uniformly n_{prop} nodes as *proposers* $\mathcal{N}_{\text{prop}}(s)$ and n_{relay} nodes as *relays* $\mathcal{N}_{\text{relay}}(s)$. For a given relay $r \in \mathcal{N}_{\text{relay}}(s)$, we use $\mathcal{N}_{\text{relay}}(s)^{-1}[r]$ to denote the index of relay r for slot s. This is relevant, as proposers send specific information to each relay based on that relay's index in a slot. The protocol operates over an underlying consensus protocol Π_{ab} which follows the enriched interface described in Sec. 2.3. The consensus protocol comes with a *leader* $\mathcal{L}(s)$ (which we assume to be sampled independently and uniformly) and a *slot proposal time* T_s for each slot s. Given the schedule T_s for Π_{ab}, we denote the start of slot s for Π_{mcp} by $\tilde{T}_s = T_s - 2\Delta$.

3.2.2 Proposer Phase (Alg. 3, ln. 8)

At time $T_s - 2\Delta$, each proposer $p \in \mathcal{N}_{\text{prop}}(s)$ assembles pending transactions into a *batch* $\boldsymbol{m}$ (Alg. 3, ln. 10).[6] The proposer then HECC encodes (using fresh randomness) this batch into n_{relay} *shreds* $\boldsymbol{c}$, where each shred c_i is intended for relay i (Alg. 3, ln. 12). This encoding preserves hiding, since no information can be learned about $\boldsymbol{m}$ from no more than τn_{relay} shreds, while γn_{relay} shreds allow reconstruction of $\boldsymbol{m}$. To ensure consistency during reconstruction (Alg. 4, ln. 34), the proposer computes a vector commitment C over the entire shred vector $\boldsymbol{c}$ using additional HECC-encoded randomness (Alg. 3, lns. 13 to 15). This ensures that the vector commitment remains hiding given the openings known to adversarial relays, but given the openings known to

[6] For the purposes of this writeup, we ignore block size limits and transaction fees. We assume batches can be arbitrarily large and include all transactions a proposer hears about. See Sec. 5.3 for an example of how to address this.

Algorithm 3 Basic multiple concurrent proposers (MCP) gadget Π_{mcp} to augment any consensus protocol Π_{ab} for combined protocol $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ to satisfy Defs. 9 and 11, with target resilience τ. **Part 1/2:** Proposer, Relay, Consensus phases. See Alg. 4 for part 2/2: Reconstruction phase. Pseudocode described from the perspective of node p.

// Parameters: coding rate $\gamma \in [0,1]$, relay threshold $\mu \in [0,1]$, availability threshold $\varphi \in [0,1]$; HECC uses parameters $N = n_{\text{relay}}$, $K = (\gamma - \tau)n_{\text{relay}}$, $T = \tau n_{\text{relay}}$.
// System-wide setup: every node p has $(\mathsf{sk}_p, \mathsf{pk}_p) \leftarrow \mathsf{Sig.Gen}(1^\kappa)$; all pk_p are common knowledge; subsequently, in the algorithm below, $\mathsf{sk} \triangleq \mathsf{sk}_p$.
// System-wide random lottery: for every slot s, sample uniformly *vectors* of n_{prop} nodes as proposers $\mathcal{N}_{\text{prop}}(s)$, and n_{relay} nodes as relays $\mathcal{N}_{\text{relay}}(s)$. Note that $\mathcal{N}_{\text{prop}}(s)$ and $\mathcal{N}_{\text{relay}}(s)$ are *ordered*, so that for $q \in \mathcal{N}_{\text{relay}}(s)$, $\mathcal{N}_{\text{relay}}(s)^{-1}[q]$ is the position of q in $\mathcal{N}_{\text{relay}}(s)$. Analogously for $\mathcal{N}_{\text{prop}}(s)$.
// Interaction with consensus protocol: for every slot s, one node is sampled independently and uniformly as leader $\mathcal{L}(s)$, and there is a slot proposal time T_s.
$L \leftarrow []$ // Initialize empty annotated output log (default value for uninitialized entries: $\perp$)
$S \leftarrow \emptyset$ // Dictionary to store shreds received when acting as a relay (default value for uninitialized entries: $\perp$)
for slot $s = 1, 2, ...$
  at $T_s - 2\Delta$ // Proposer phase
    if $p \in \mathcal{N}_{\text{prop}}(s)$ // Node p is a proposer for slot s
      Collect pending transactions (from invocations of $\Pi_{\text{mcp}}(\Pi_{\text{ab}}).\mathsf{Input}(s, \mathsf{txs})$) into batch $\boldsymbol{m}$
      $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^T$ // Sample randomness for HECC
      $\boldsymbol{c} = (c_1, ..., c_{n_{\text{relay}}}) \leftarrow \mathsf{HECC.Enc}(\boldsymbol{m}, \boldsymbol{r})$
      $\boldsymbol{m}' \xleftarrow{\$} \mathbb{F}^K$, $\boldsymbol{r}' \xleftarrow{\$} \mathbb{F}^T$ // Sample randomness for commitment hiding
      $\boldsymbol{r} = (r_1, ..., r_{n_{\text{relay}}}) \leftarrow \mathsf{HECC.Enc}(\boldsymbol{m}', \boldsymbol{r}')$
      $C \leftarrow \mathsf{VC.Commit}(\boldsymbol{c}, \boldsymbol{r})$
      for $r \in \mathcal{N}_{\text{relay}}(s)$
        $w_r \leftarrow \mathsf{VC.Open}(\boldsymbol{c}, \boldsymbol{r}, \mathcal{N}_{\text{relay}}(s)^{-1}[r])$ // Recall, $\mathcal{N}_{\text{relay}}(s)^{-1}[q]$ is the position of q in $\mathcal{N}_{\text{relay}}(s)$.
        $\sigma_r \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, C)$
        Privately send $(C, c_{\mathcal{N}_{\text{relay}}(s)^{-1}[r]}, r_{\mathcal{N}_{\text{relay}}(s)^{-1}[r]}, w_r, \sigma_r)$ to relay r
  at $T_s - \Delta$ // Relay phase
    if $p \in \mathcal{N}_{\text{relay}}(s)$ // Node p is a relay for slot s
      $A \leftarrow \emptyset$ // Initialize empty dictionary
      for $q \in \mathcal{N}_{\text{prop}}(s)$
        if $(C_q, c_q, r_q, w_q, \sigma_q)$ privately received from q
          assert $\mathsf{Sig.Verify}(\mathsf{pk}_q, C_q, \sigma_q) \wedge \mathsf{VC.Verify}(C_q, \mathcal{N}_{\text{relay}}(s)^{-1}[p], c_q, r_q, w_q)$
          $S[s, q] \leftarrow (C_q, c_q, r_q, w_q, \sigma_q)$ // Store shred and randomness for proposer q in slot s
          $A[q] \leftarrow (C_q, \sigma_q)$ // Attest to having stored shred for proposer q in slot s
      $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, A)$
      Send (A, σ) to consensus leader $\mathcal{L}(s)$
  at T_s // Consensus phase
    if $p = \mathcal{L}(s)$ // Node p is the consensus leader for slot s
      $B \leftarrow \emptyset$ // Initialize empty dictionary
      for $r \in \mathcal{N}_{\text{relay}}(s)$
        if (A_r, σ_r) received from r
          assert $\mathsf{Sig.Verify}(\mathsf{pk}_r, A_r, \sigma_r)$
          for $(q \mapsto (C_q, \sigma_q)) \in A_r$
            assert $(q \in \mathcal{N}_{\text{prop}}(s)) \wedge \mathsf{Sig.Verify}(\mathsf{pk}_q, C_q, \sigma_q)$
          $B[r] \leftarrow (A_r, \sigma_r)$
      $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, B)$
      Invoke $\Pi_{\text{ab}}.\mathsf{Input}(s, (B, \sigma))$ // Input to consensus protocol
  // See Alg. 4 for part 2/2: Reconstruction phase.

Algorithm 4 Basic multiple concurrent proposers (MCP) gadget Π_{mcp} to augment any consensus protocol Π_{ab} for combined protocol $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ to satisfy Defs. 9 and 11, with target resilience τ. **Part 2/2:** RECONSTRUCTION phase. See Alg. 3 for part 1/2: PROPOSER, RELAY, CONSENSUS phases. Pseudocode described from the perspective of node p.

1 // See Alg. 3 for parameters and initialization.
2 // Messages received as part of the RECONSTRUCTION phase are automatically re-broadcast to all nodes.
3 **for** slot $s = 1, 2, ...$
4 // See Alg. 3 for part 1/2: PROPOSER, RELAY, CONSENSUS phases.
5 **upon** $\Pi_{\text{ab}}.\mathsf{Output}(s, \perp)$ triggered and $L[s-1] \neq \perp$
6 $L[s] \leftarrow \emptyset$, triggering $\Pi_{\text{mcp}}(\Pi_{\text{ab}}).\mathsf{Output}(s, L[s])$
7 **upon** $\Pi_{\text{ab}}.\mathsf{Output}(s, (B, \sigma))$ triggered and $L[s-1] \neq \perp$ // RECONSTRUCTION PHASE
8 **try**
9 **assert** $\mathsf{Sig.Verify}(\mathsf{pk}_{\mathcal{L}(s)}, B, \sigma) \wedge (|B| \geq \mu n_{\text{relay}})$
10 **for** $(r \mapsto (A_r, \sigma_r)) \in B$
11 **assert** $(r \in \mathcal{N}_{\text{relay}}(s)) \wedge \mathsf{Sig.Verify}(\mathsf{pk}_r, A_r, \sigma_r)$
12 **for** $(q \mapsto (C_q, \sigma_q)) \in A_r$
13 **assert** $(q \in \mathcal{N}_{\text{prop}}(s)) \wedge \mathsf{Sig.Verify}(\mathsf{pk}_q, C_q, \sigma_q)$
14 **catch** assertion failure
15 $L[s] \leftarrow \emptyset$, triggering $\Pi_{\text{mcp}}(\Pi_{\text{ab}}).\mathsf{Output}(s, L[s])$
16 **break** out of ln. 7 "upon" block
17 $P \leftarrow \emptyset$ // Set of proposers whose batches to reconstruct for $L[s]$
18 **for** $q \in \mathcal{N}_{\text{prop}}(s)$
19 $A' \leftarrow \{(r, C) \mid \exists A, \sigma, \sigma' : (B[r] = (A, \sigma)) \wedge (A[q] = (C, \sigma'))\}$ // Relay attestations for q
20 **if** $(|A'| \geq \varphi n_{\text{relay}}) \wedge \exists C : \forall (r, C') \in A' : C' = C$ // No equivocation, enough shreds available
21 $P \leftarrow P \cup \{(q, C)\}$
22 **if** $p \in \mathcal{N}_{\text{relay}}(s)$ // Node p is a relay for slot s
23 **for** $(q, C) \in P$
24 Broadcast $(s, p, q, S[s, q])$ // Broadcast shred stored for proposer q in slot s (if any)
25 $S_s \leftarrow \emptyset$ // Initialize empty dictionary
26 **upon** $(s, p, q, (C, c, r, w, \sigma))$ received // Collect shreds for proposers whose batches to reconstruct for $L[s]$
27 **assert** $(q, C) \in P \wedge \mathsf{Sig.Verify}(\mathsf{pk}_q, C, \sigma) \wedge \mathsf{VC.Verify}(C, \mathcal{N}_{\text{relay}}(s)^{-1}[p], c, r, w)$
28 $S_s[p, q] \leftarrow (c, r)$
29 **upon** $\forall (q, C) \in P : |\{p \mid S_s[p, q] \neq \perp\}| \geq \gamma n_{\text{relay}}$ // Enough shreds collected to reconstruct batches
30 $\boldsymbol{m} \leftarrow \emptyset$ // Initialize empty dictionary
31 **for** $(q, C) \in P$
32 $(\boldsymbol{m}[q], \boldsymbol{r}[q]) \leftarrow \mathsf{HECC.Dec}(\{(S_s[p, q][0], \mathcal{N}_{\text{relay}}(s)^{-1}[p]) \mid S_s[p, q] \neq \perp\})$
33 $(\boldsymbol{m}'[q], \boldsymbol{r}'[q]) \leftarrow \mathsf{HECC.Dec}(\{(S_s[p, q][1], \mathcal{N}_{\text{relay}}(s)^{-1}[p]) \mid S_s[p, q] \neq \perp\})$
34 **if** $\mathsf{VC.Commit}(\mathsf{HECC.Enc}(\boldsymbol{m}[q], \boldsymbol{r}[q]), \mathsf{HECC.Enc}(\boldsymbol{m}'[q], \boldsymbol{r}'[q])) \neq C$
35 $\boldsymbol{m}[q] \leftarrow \perp$
36 $L[s] \leftarrow \text{orderAndConcat}(\boldsymbol{m})$, triggering $\Pi_{\text{mcp}}(\Pi_{\text{ab}}).\mathsf{Output}(s, L[s])$ // Order transactions according to application logic; output annotated log

honest relays, both the underlying vector and randomness can be reconstructed. For each relay $r \in \mathcal{N}_{\text{relay}}(s)$, the proposer generates an opening proof w_r demonstrating that shred $c_{\mathcal{N}_{\text{relay}}(s)^{-1}[r]}$ appears at index $\mathcal{N}_{\text{relay}}(s)^{-1}[r]$ within the committed vector (Alg. 3, ln. 17) together with the commitment randomness $r_{\mathcal{N}_{\text{relay}}(s)^{-1}[r]}$. The proposer signs the commitment C producing the signature σ_r (Alg. 3, ln. 18), and sends the shred packet $(C, c_{\mathcal{N}_{\text{relay}}(s)^{-1}[r]}, r_{\mathcal{N}_{\text{relay}}(s)^{-1}[r]}, w_r, \sigma_r)$ privately to relay r (Alg. 3, ln. 19).

3.2.3 Relay Phase (Alg. 3, ln. 20)

At time $T_s - \Delta$, each relay $p \in \mathcal{N}_{\text{relay}}(s)$ verifies shreds received from proposers. Specifically, for each proposer q that transmitted a shred packet $(C_q, c_q, r_q, w_q, \sigma_q)$, the relay verifies both the proposer's signature σ_q on commitment C_q and the opening proof w_q demonstrating that c_q appears at the expected position $\mathcal{N}_{\text{relay}}(s)^{-1}[p]$ in the committed shred vector, associated with randomness r_q (Alg. 3, ln. 25). This ensures that adversarial nodes cannot forge shred packets, and enables consistency during reconstruction (Alg. 4, ln. 34). The relay stores (Alg. 3, ln. 26) each verified shred packet in local memory $S[s, q]$ for potential later broadcast (Alg. 4, ln. 24), but, crucially, does not reveal the shred at this point. Instead, the relay constructs an *attestation* A, a dictionary mapping q to (C_q, σ_q) (Alg. 3, ln. 27). This attestation certifies that the relay received, verified, and stored a shred from proposer q, while revealing only the commitment C_q and the signature σ_q, which reveal no information about the individual shred c_q due to the hiding property of the vector commitment scheme. The relay signs attestation A (Alg. 3, ln. 28), and sends the signed attestation (A, σ) to the consensus leader $\mathcal{L}(s)$ of slot s (Alg. 3, ln. 29).

3.2.4 Consensus Phase (Alg. 3, ln. 30)

At time T_s, the *leader* $\mathcal{L}(s)$ for slot s of the consensus protocol Π_{ab} collects valid attestations from the relays, and assembles a *block* B for input to Π_{ab}. Specifically, the leader validates each attestation (A_r, σ_r) received from relay r by verifying the relay's signature σ_r on A_r (Alg. 3, ln. 35), and by verifying the proposer's signature within each entry $(q \mapsto (C_q, \sigma_q)) \in A_r$ (Alg. 3, ln. 37). Successfully validated attestations are incorporated into block B as $B[r]$ (Alg. 3, ln. 38). The leader signs B (Alg. 3, ln. 39), and submits the signed block as input (B, σ) for slot s to the underlying consensus protocol Π_{ab} (Alg. 3, ln. 40).

3.2.5 Reconstruction Phase (Alg. 4, ln. 7)

Upon receiving a signed block (B, σ) from Π_{ab} as consensus decision for slot s, all nodes initiate reconstruction (Alg. 4, ln. 7). Nodes first validate (B, σ) by verifying the leader's signature σ and confirming that B contains attestations from at least μn_{relay} relays (Alg. 4, ln. 9)—a threshold that will be tuned in Sec. 4 appropriately to ensure selective-censorship resistance. Each embedded attestation (A_r, σ_r) undergoes signature verification, as do the proposers' signatures on (C_q, σ_q) within each attestation A_r (Alg. 4, ln. 11, Alg. 4, ln. 13). Note that all checks are deterministic and thus all honest nodes reach the same verdict. If any check fails, the block is rejected and an empty consensus decision is returned by $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ for slot s (Alg. 4, ln. 15).

Nodes determine which proposers' batches are available for reconstruction and required to be included in the consensus decision of $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ for slot s, by analyzing the relays' attestations. For each proposer $q \in \mathcal{N}_{\text{prop}}(s)$, nodes collect all attestations $A_r[q]$ where A_r is relay r's attestation, for all relays r where relay r attested to having received a valid shred for proposer q (Alg. 4, ln. 19). The batch of proposer q is deemed *available* (Alg. 4, ln. 21) if at least φn_{relay} relays attested to the same commitment C and no relay attested to a conflicting commitment (Alg. 4, ln. 20). The availability threshold φ will be tuned appropriately in Sec. 4 to ensure that the proposer's batch can be reconstructed from the shreds stored by honest relays for that batch. Note again that all computations are deterministic, and honest nodes have consensus on (B, σ), and thus all honest nodes reach the same verdict regarding which proposers' batches are available.

Note that only after consensus has been reached on which proposers' batches are to be included in the consensus decision of $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ for slot s, do relays broadcast their stored shred packets $S[s, q]$ for each proposer q whose batch was deemed available (Alg. 4, ln. 24). This order of events preserves hiding, as the adversary learns no information about honest proposers' batches from the shreds stored at the roughly fn_{relay}/n adversarial relays, due to the hiding property of the HECC and of the vector commitment. Nodes collect broadcast shred packets, and, upon receiving at least γn_{relay} valid shreds and associated randomness values for proposer q (Alg. 4, ln. 29), apply HECC decoding to reconstruct both the original batch $\boldsymbol{m}_q$ and its encoding randomness $\boldsymbol{r}_q$, as well as the commitment randomness (Alg. 4, lns. 32 and 33). Consistency of reconstruction is ensured by re-encoding the recovered batch and commitment randomness, and checking that the resulting vector commitment matches the attested C (Alg. 4, ln. 34). If not, which happens exclusively if an adversary proposer does not follow the HECC encoding and vector commitment scheme during the proposer phase, the batch is discarded (Alg. 4, ln. 35). Valid reconstructed batches are added to the annotated consensus output log $L[s]$ of $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ for slot s according to a specific deterministic transaction ordering rule (Alg. 4, ln. 36).

4 Analysis

4.1 Parameter Choices

Regardless of our parameter choices for γ, φ, μ in Alg. 3, as long as Π_{ab} is safe against f Byzantine nodes, $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ is also safe against f Byzantine nodes. However, the other three properties of liveness, selective-censorship resistance, and hiding only hold under specific choices of μ (relay threshold: minimum relay attestations required for a valid consensus block), φ (availability threshold: minimum attestations to attempt batch reconstruction), γ (coding rate: fraction of shreds needed for reconstruction), in relation to the resilience $\tau \triangleq f/n$ of Π_{ab}, which also determines T (HECC parameter such that at most T shreds reveal no information about batches' contents). Let f_{relay} be the number of Byzantine relay nodes in a given slot. We list the inequalities required so that our subsequent analyses of the security properties of $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ hold.

Liveness. $f_{\text{relay}} \leq (\varphi - \gamma)n_{\text{relay}}$. $\tau \leq \varphi - \gamma$. For a liveness violation to occur, it must be that a batch is considered available (received φn_{relay} relay attestations) but less than γn_{relay} relays release shreds for the batch, so that nodes are unable to complete reconstruction.[7] The condition ensures that batches deemed available will eventually be reconstructed by all honest nodes.

Selective-Censorship Resistance. $f_{\text{relay}} \leq (\mu - \varphi)n_{\text{relay}}$. $\tau \leq \mu - \varphi$. For an adversary to selectively censor a transaction, they must control enough relays that do not attest to the corresponding batches in which that transaction is included, so that none of those batches are considered available, even though a valid block is proposed by a potentially Byzantine leader. The condition ensures that a valid block must contain enough honest relay attestations to deem all batches of honest proposers available.

Hiding. $f_{\text{relay}} \leq T$. $\tau \leq T/n_{\text{relay}}$. Honest relays only release shreds as part of $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ after a batch is considered available and confirmed by Π_{ab}. Thus, the only information an adversary has about transactions included in batches deemed available, is from the shreds received by Byzantine relays. The condition ensures that from those shreds, an adversary cannot learn any information

[7] See Sec. 5.2 for an additional gadget that allows to recover liveness if such an event occurs.

about batches of honest proposers.

Note that increasing μ, and decreasing γ towards T, weakly improves the resilience of all three of these properties. At the same time, $f_{\text{relay}} \leq (1-\mu)n_{\text{relay}}$ must hold, otherwise adversarial relays withholding attestations can keep honest consensus leaders from proposing valid blocks. Furthermore, smaller γ means less of HECC encoding is used for payload vs. randomness, causing net transaction throughput to go down. Furthermore, $\gamma n_{\text{relay}} = K + T$, so $\gamma n_{\text{relay}} > T$ is required.

Concretely, we instantiate $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ with parameters (Alg. 3, ln. 1) $\mu = 4/5$, $\varphi = 3/5$, and $\gamma = 2/5$, for a target resilience of $\tau = 1/5$. We assume $n_{\text{prop}} = n$ and $n_{\text{relay}} = \Theta(n)$. That is, we assume that, in each slot, all nodes are eligible to submit batches of transactions, and a constant fraction of the nodes are relays sampled uniformly (Alg. 3, ln. 3). Consequently, HECC is instantiated with $N = n_{\text{relay}}$, $K = (\gamma - \tau)n_{\text{relay}} = n_{\text{relay}}/5$, and $T = \tau n_{\text{relay}} = n_{\text{relay}}/5$, so that at most T shreds reveal no information about batches' contents, and $K + T = 2n_{\text{relay}}/5 = \gamma n_{\text{relay}}$ shreds suffice to reconstruct batches (Alg. 4, ln. 32). We also assume that the component protocol Π_{ab} is secure (Def. 7) with resilience 1/5 and satisfies the enriched consensus interface of Sec. 2.3.2 (Alg. 3, ln. 4) and leader-driven liveness (Def. 8).

Definition 12. A sequence of events $\{E_n\}_{n\in\mathbb{N}}$ occurs *with overwhelming probability* (w.o.p.) iff the probability of the sequence of complementary events $\{\overline{E_n}\}_{n\in\mathbb{N}}$ is negligible in n.

We start by showing that w.o.p., for every slot s, at least $4n_{\text{relay}}/5$ of the relays are honest.

Lemma 5. *Over execution horizons of* $\text{poly}(n)$ *slots, w.o.p.,* $\forall s : |\mathcal{N}_{\text{relay}}(s) \cap \mathcal{N}_{\text{a}}| < n_{\text{relay}}/5$.

Proof of Lem. 5. Let X_s denote the number of Byzantine relays in slot s. Since the adversary controls at most $f = \beta n$ nodes where $\beta < 1/5$, and relays are randomly selected, we have $\mathbb{E}[X_s] = \beta n_{\text{relay}}$. For any fixed slot s, applying a Chernoff bound yields

$$\mathbb{P}\left[X_s \geq \frac{n_{\text{relay}}}{5}\right] = \mathbb{P}\left[X_s \geq (1+\delta)\mathbb{E}[X_s]\right] \leq \exp\left(-\frac{\beta n_{\text{relay}}\delta^2}{2+\delta}\right) = \exp\left(-\frac{n_{\text{relay}}(1-5\beta)^2}{5(1+5\beta)}\right)$$

where $\delta = \frac{1}{5\beta} - 1$ results from $\mathbb{E}[X_s] = \beta n_{\text{relay}}$ and setting $(1+\delta)\beta n_{\text{relay}} = n_{\text{relay}}/5$.

Since $\beta < 1/5$ is constant and $n_{\text{relay}} = \Theta(n)$, this probability is $\exp(-\Theta(n))$. Taking a union bound over $\text{poly}(n)$ slots, the probability that any slot has $n_{\text{relay}}/5$ or more Byzantine relays is at most $\text{poly}(n) \cdot \exp(-\Theta(n)) = \text{negl}(n)$. □

For the rest of the analysis, we assume that every slot has at least $4n_{\text{relay}}/5$ honest relays, except for the safety of the protocol, which is satisfied even when this assumption is not met.

4.2 Safety

We start by showing that $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ is safe. Intuitively, the safety of the underlying consensus protocol Π_{ab} implies that every honest node has the same view of what batches are deemed available and of the vector commitments to the shreds each proposer produced for their batches. We show that by using these vector commitments, either all honest nodes give up on a particular proposer's batch, or they reconstruct the same batch. From this we get that all honest nodes must reconstruct the same set of batches for every slot. Since the output log of transactions for a slot is a deterministic function of the reconstructed batches (Alg. 4, ln. 36), this implies honest nodes always have consistent logs.

Note that it is possible (albeit under appropriate parameters, very unlikely) for the protocol to subsample a set of relays where more than $n_{\text{relay}}/5$ relays are Byzantine. In this case, the adversarial relays can withhold shreds they attested to receiving, causing honest nodes to be unable to reconstruct the batches deemed available. In this case, honest nodes are caught waiting for these shreds, and liveness is violated. However, safety is not harmed by this. Sec. 5.2 discusses a technique to resolve consensus stalls in those cases.

Theorem 1. *Assuming* Π_{ab} *is secure with resilience* $n/5$, $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ *is safe (Def. 5).*

Proof sketch of Thm. 1. Fix any slot s and any two honest nodes p_1, p_2. We show that if p_1 eventually assigns $L^{p_1}[s] = m$, then, if p_2 assigns $L^{p_2}[s]$, then $L^{p_2}[s] = m$. Observe from Alg. 4 that honest nodes assign a slot $L[s]$ only once, and only in strictly increasing order after $L[s-1]$ is assigned. Let B be the output of Π_{ab} for slot s. By safety of Π_{ab}, all honest nodes, especially p_1 and p_2, receive the same B for slot s from Π_{ab}.

If $B = \perp$, or B does not pass verification (Alg. 4, lns. 9, 11 and 13), then every honest node p assigns $L^p[s] = \emptyset$ (Alg. 4, lns. 6 and 15), as desired. Thus consider the case that B is a valid block passing verification. Let P be the set of proposers whose batches are deemed available by B according to the availability condition (Alg. 4, ln. 20). Note that P is a deterministic function of B, so all honest nodes have the same view of P. We show that for any proposer q such that $(q, C) \in P$, regardless of what set of shreds any honest node receives, either all nodes reconstruct the same batch $\boldsymbol{m}[q]$, or all nodes fail to verify that the HECC encoding of the batch they reconstruct matches the vector commitment found in B, and hence set $\boldsymbol{m}[q] = \perp$.

Consider a pair $(q, C) \in P$ in Alg. 4, ln. 31, and the two honest nodes p_1 and p_2. Note that this implies that both p_1 and p_2 each have received γn_{relay} valid shreds for the proposer q from the relays (Alg. 4, ln. 29). Note that for every shred packet $(s, p, q, (C, c, r, w, \sigma))$ received by an honest node (Alg. 4, ln. 27), the node checks w is a valid opening of C to (c, r) at index $\mathcal{N}_{\text{relay}}(s)^{-1}[p]$. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ denote the sets of relays from which p_1 and p_2, respectively, received valid shreds about the proposal of q. Let S_1 and S_2 denote the dictionaries p_1 and p_2 store for these shreds for slot s, and let $(\boldsymbol{m}_1[q], \boldsymbol{r}_1[q], \boldsymbol{m}'_1[q], \boldsymbol{r}'_1[q]), (\boldsymbol{m}_2[q], \boldsymbol{r}_2[q], \boldsymbol{m}'_2[q], \boldsymbol{r}'_2[q])$ denote the messages p_1 and p_2 decode, respectively, from S_1 and S_2 as defined in Alg. 4, lns. 32 and 33.

Let

$$\tilde{\boldsymbol{v}}_k = (\tilde{v}_{k,1}, ..., \tilde{v}_{k,N}) \triangleq \mathsf{HECC.Enc}(\boldsymbol{m}_k[q], \boldsymbol{r}_k[q])$$

and

$$\tilde{\boldsymbol{r}}_k = (\tilde{r}_{k,1}, ..., \tilde{r}_{k,N}) \triangleq \mathsf{HECC.Enc}(\boldsymbol{m}'_k[q], \boldsymbol{r}'_k[q])$$

for $k \in \{1, 2\}$. We first show that $\mathsf{VC.Commit}(\tilde{\boldsymbol{v}}_1, \tilde{\boldsymbol{r}}_1) = C$ iff $\mathsf{VC.Commit}(\tilde{\boldsymbol{v}}_2, \tilde{\boldsymbol{r}}_2) = C$ in Alg. 4, ln. 34. Without loss of generality, assume $\mathsf{VC.Commit}(\tilde{\boldsymbol{v}}_1, \tilde{\boldsymbol{r}}_1) = C$. By the position binding property of VC (Def. 2), since $\mathsf{VC.Commit}(\tilde{\boldsymbol{v}}_1, \tilde{\boldsymbol{r}}_1) = C$, and since p_2 has verified the openings of the shreds it received, it must be the case that for all $p \in \mathcal{I}_2$, $S_2[p, q][0] = \tilde{v}_{1,\mathcal{N}_{\text{relay}}(s)^{-1}[p]}$ and $S_2[p, q][1] = \tilde{r}_{1,\mathcal{N}_{\text{relay}}(s)^{-1}[p]}$. Then, by the erasure-correction property of HECC (Def. 4), it must be the case that

$$(\boldsymbol{m}_1[q], \boldsymbol{r}_1[q], \boldsymbol{m}'_1[q], \boldsymbol{r}'_1[q]) = (\boldsymbol{m}_2[q], \boldsymbol{r}_2[q], \boldsymbol{m}'_2[q], \boldsymbol{r}'_2[q]).$$

Since $\mathsf{HECC.Enc}$ is deterministic, this implies that $\tilde{\boldsymbol{v}}_1 = \tilde{\boldsymbol{v}}_2$ and $\tilde{\boldsymbol{r}}_1 = \tilde{\boldsymbol{r}}_2$, and since $\mathsf{VC.Commit}$ is deterministic, this implies that $\mathsf{VC.Commit}(\tilde{\boldsymbol{v}}_1, \tilde{\boldsymbol{r}}_1) = \mathsf{VC.Commit}(\tilde{\boldsymbol{v}}_2, \tilde{\boldsymbol{r}}_2) = C$, as desired.

It is clear that p_1 overwrites $\boldsymbol{m}[q] \leftarrow \perp$ in Alg. 4, ln. 35 iff p_2 does the same. It remains to show that if neither p_1 nor p_2 overwrite $\boldsymbol{m}[q] \leftarrow \perp$ in Alg. 4, ln. 35, then $\boldsymbol{m}_1[q] = \boldsymbol{m}_2[q]$. This follows

from observing that under those circumstances, $\mathsf{VC.Commit}(\tilde{\boldsymbol{v}}_1, \tilde{\boldsymbol{r}}_1) = C = \mathsf{VC.Commit}(\tilde{\boldsymbol{v}}_2, \tilde{\boldsymbol{r}}_2)$, so that by the binding property of VC (Def. 2), $\tilde{\boldsymbol{v}}_1 = \tilde{\boldsymbol{v}}_2$ and $\tilde{\boldsymbol{r}}_1 = \tilde{\boldsymbol{r}}_2$, and thus by the erasure-correction property of HECC (Def. 4), $\boldsymbol{m}_1[q] = \boldsymbol{m}_2[q]$, as desired.

Since every honest node waits to reconstruct the batches from every proposer in P before assigning anything to $L[s]$ in Alg. 4, ln. 36, and for every proposer the corresponding batch is reconstructed identically across honest nodes, and the resulting transactions from the reconstructed batches are ordered deterministically, we have that any two honest nodes that assign anything to $L[s]$ in Alg. 4, ln. 36, must assign the same contents. □

4.3 Selective-Censorship Resistance

Before showing that $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ is live, we first show that $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ is selective-censorship resistant. Liveness then follows from $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ being selective-censorship resistant and Π_{ab} additionally satisfying leader-driven liveness. Intuitively, there are two ways for an adversary to potentially selectively censor a transaction. The adversary can control enough relays and have them claim not to have seen a batch from a certain proposer; or a Byzantine consensus leader can purposefully leave out relay attestations corresponding to a target batch. We show that given the adversary controls fewer than $n_{\text{relay}}/5$ relays, neither attack (nor any other attack) works.

As a result, an adversary's only choice, if they wish to censor any target transactions, is to control the consensus leader for that slot *and* either assemble an invalid block or not assemble any block at all. While this gives an adversary *some* ability to censor, an adversary cannot include any of their own transactions either, and everyone has the ability to resubmit fresh transactions in the next slot. In other words, while the adversary can censor *indiscriminately*, it cannot do so *selectively*, which is precisely what Def. 9 asserts.

Theorem 2. *Assuming Π_{ab} is secure with resilience $n/5$, w.o.p., $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ is selective-censorship resistant (Def. 9).*

Proof sketch of Thm. 2. Following Lem. 5, we consider only executions where $\forall s : f_{\text{relay}} \triangleq |\mathcal{N}_{\text{relay}}(s) \cap \mathcal{N}_{\text{a}}| < n_{\text{relay}}/5$, and, following Thm. 1, we may assume $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ is safe. Consider a slot s for which $\tilde{T}_s \geq \mathsf{GST}$. Recall $\tilde{T}_s = T_s - 2\Delta$ where T_s refers to the time when slot s starts under Π_{ab}. Let a transaction tx arrive at an honest node p before time $\tilde{T}_s$. We show that then either $\mathsf{tx} \in L^p_\infty[s]$ or $L^p_\infty[s] = \emptyset$ (and so, by safety of $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$, either $\mathsf{tx} \in L^*_\infty[s]$ or $L^*_\infty[s] = \emptyset$, as required).

If tx has not yet been included in any confirmed slot, then honest node p includes tx in its batch $\boldsymbol{m}$ for slot s (Alg. 3, ln. 10). Then p samples randomness $\boldsymbol{r}$ (Alg. 3, ln. 11), encodes the batch into shreds $\boldsymbol{c} = \mathsf{HECC.Enc}(\boldsymbol{m}, \boldsymbol{r})$ (Alg. 3, ln. 12), and distributes these shreds with vector commitments and opening proofs to the relays (Alg. 3, ln. 19). Since honest nodes broadcast their shreds at $\tilde{T}_s$, and $\tilde{T}_s \geq \mathsf{GST}$, every honest relay r will receive their shred and corresponding vector commitment randomness along with valid opening proofs by $T_s - \Delta$. It follows that every honest relay attests to the proposal of p being available (Alg. 3, ln. 27) and gets their attestation to the consensus leader $\mathcal{L}(s)$ by T_s (Alg. 3, ln. 29).

By the liveness of Π_{ab}, eventually, all nodes receive either $\Pi_{\text{ab}}.\mathsf{Output}(s, \perp)$ or $\Pi_{\text{ab}}.\mathsf{Output}(s, (B, \sigma))$. In the former case, every honest node p outputs $L^p_\infty[s] = \emptyset$ (Alg. 4, ln. 6). In the latter case, if B is an invalid block, all honest nodes also output $L^p_\infty[s] = \emptyset$ (Alg. 4, ln. 15). Otherwise, if B is a valid block, it must contain at least μn_{relay} unique relay attestations (Alg. 4, ln. 9). Since

at least $(\mu - 1/5)n_{\text{relay}} \geq \varphi n_{\text{relay}}$ of these attestations are from honest relays, we get that the proposal of p must be deemed available by B (Alg. 4, ln. 20). By the safety of Π_{ab}, all honest relays have the same view of B and thus of P (Alg. 4, ln. 21), and hence will all release their shreds for the proposal of p to all other nodes (Alg. 4, ln. 24). Thus, all honest nodes receive at least $\varphi n_{\text{relay}} \geq \gamma n_{\text{relay}}$ shreds for the proposal of p (Alg. 4, ln. 29), and will successfully reconstruct $\boldsymbol{m}$ from decoding the shreds (Alg. 4, ln. 32). Since the union of all transactions in reconstructed batches for slot s gets added to the output log for slot s (Alg. 4, ln. 36), we have $\mathsf{tx} \in L^p_\infty[s]$ for every honest node p. □

4.4 Liveness

The liveness of MCP follows similarly, except we consider a slot s with $\tilde{T}_s \geq \mathsf{GST} + c\Delta$, where c is the constant of the leader-driven liveness guarantee (Def. 8) of Π_{ab}, and $\mathcal{L}(s) \in \mathcal{N}_{\text{h}}$.

Theorem 3. *Assuming Π_{ab} is secure with resilience $n/5$ and satisfies leader-driven liveness (Def. 8), then, w.o.p., $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ is live (Def. 6) (and even satisfies leader-driven liveness).*

Proof sketch of Thm. 3. Following Lem. 5, we consider only executions where $\forall s : f_{\text{relay}} \triangleq |\mathcal{N}_{\text{relay}}(s) \cap \mathcal{N}_{\text{a}}| < n_{\text{relay}}/5$.

We first show that for every s, every honest node p eventually completes the reconstruction phase (Alg. 4) and assigns $L^p[s]$. We proceed by induction. The claim is vacuously true for $s = 0$. Assume p eventually completes the reconstruction phase for slot $k-1$. Then, for slot k, p is only blocked on receiving $\Pi_{\text{ab}}.\mathsf{Output}(k, \perp)$ or $\Pi_{\text{ab}}.\mathsf{Output}(k, (B, \sigma))$ to start reconstruction. Since Π_{ab} satisfies leader-driven liveness, p eventually witnesses one of these events. If $\Pi_{\text{ab}}.\mathsf{Output}(k, \perp)$ (Alg. 4, ln. 6) or $\Pi_{\text{ab}}.\mathsf{Output}(k, (B, \sigma))$ where B is deemed invalid (Alg. 4, lns. 9, 11 and 13), then p assigns $L^p[k] = \emptyset$, as desired.

Otherwise, let P be the set of proposers whose batches are deemed available by B, according to the availability condition of Alg. 4, ln. 20. Note that P is a deterministic function of B, so all honest nodes have identical P, as shown in the proof of Thm. 1. Since any $(q, C) \in P$ has φn_{relay} corresponding attestations in B, we have that at least $\varphi n_{\text{relay}} - f_{\text{relay}} \geq \gamma n_{\text{relay}}$ honest relays attested to (q, C) and custody valid shreds for the batch of proposer q. Thus, each of these honest relays will eventually release their shred, and for every $(q, C) \in P$, every honest node p will receive at least γn_{relay} valid shreds, allowing them to complete the reconstruction phase and assign $L^p[s]$ (Alg. 4, ln. 36).

Now, consider a transaction tx submitted to an honest node p at time t. Then, let s be the earliest slot such that $\tilde{T}_s \geq t$, $\tilde{T}_s \geq \mathsf{GST} + c\Delta$, and $\mathcal{L}(s)$ is an honest node. (To conclude leader-driven liveness of $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$, consider $p = \mathcal{L}(s)$.) Then, at time $\tilde{T}_s$, if $\mathsf{tx} \notin L^p_{\tilde{T}_s}$, node p includes tx in its batch $\boldsymbol{m}$ (Alg. 3, ln. 10) and gets her shreds to each relay by time $T_s - \Delta$ (Alg. 3, ln. 19). All honest relays will then attest to the proposal of p being available (Alg. 3, ln. 27), and get their attestations to $\mathcal{L}(s)$ by T_s (Alg. 3, ln. 29). Since $\mathcal{L}(s)$ is honest, they will include all timely attestations in their block B (Alg. 3, ln. 38).

Since $\tilde{T}_s \geq \mathsf{GST}$, $\mathcal{L}(s)$ will hear the attestations from all honest relays in time. Since $n_{\text{relay}} - f_{\text{relay}} \geq \mu n_{\text{relay}}$, $\mathcal{L}(s)$ hears enough attestations to input a valid block B to Π_{ab} (Alg. 3, ln. 40). Then, since Π_{ab} satisfies leader-driven liveness, we have that, eventually, B gets confirmed by Π_{ab} with output $\Pi_{\text{ab}}.\mathsf{Output}(s, (B, \sigma))$. From here, the reconstruction phase for slot s is guaranteed to terminate, as discussed above, and the proof follows identically to the proof for showing $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$

is selective-censorship resistant (see proof of Thm. 2), in that B deems the proposal of p available, sufficiently many honest relays release shreds for the proposal of p (Alg. 4, ln. 24), and subsequently all honest nodes reconstruct $\boldsymbol{m}$ (Alg. 4, ln. 32), and confirm the corresponding transactions in their logs (Alg. 4, ln. 36). □

4.5 Hiding

Intuitively, since we use a HECC (Def. 3) with parameter $T = \tau n_{\text{relay}}$, along with a hiding vector commitment (Def. 2), for any slot where fewer than τn_{relay} relays are Byzantine, via Lem. 4, we have that the adversary cannot recover any information about what will become the content of $L^*_\infty[s]$ until at least one honest relay releases their shreds. However, since an honest relay only releases its shreds once the block for that slot is confirmed in Π_{ab}, an adversary only sees the contents of transactions in a slot once that slot's valency of $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ has already been determined.

Theorem 4. *Assuming Π_{ab} is secure with resilience $n/5$ and satisfies leader-driven liveness, and* HECC *is instantiated with $K = T = n_{\text{relay}}/5$ and $N = n_{\text{relay}}$ as in Sec. 4.1, w.o.p., $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ is hiding (Def. 11).*

Proof sketch of Thm. 4. Following Lem. 5, we consider only executions where $\forall s : f_{\text{relay}} \triangleq |\mathcal{N}_{\text{relay}}(s) \cap \mathcal{N}_{\text{a}}| < n_{\text{relay}}/5$. Consider the hiding game of Def. 11. The adversary $\mathcal{A}$ chooses an honest node p and transactions $\mathsf{tx}_0, \mathsf{tx}_1$. The challenger flips a local coin $b \xleftarrow{\$} \{0,1\}$ uniformly, and inputs tx_b to p. Let t^* be the first time such that $\mathsf{tx}_b \in L^{\times}_{t^*}$, where $L^{\times}_{t^*}$ denotes the valency of $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ at t^* (Def. 10).

Since p is honest and $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ is live (by Thm. 3), tx_b will eventually be confirmed, and thus t^* is well-defined and finite. Let $\boldsymbol{m}_{p,s}$ denote the batch p submits for slot s (Alg. 3, ln. 10). Note that p might have to submit batches including tx_b across multiple slots, either during periods of asynchrony before GST, or in the case that the adversary controls the consensus leaders for these slots and causes them to not propose valid blocks.

Then, let s^* be the first slot such that $\mathsf{tx}_b \in \boldsymbol{m}_{p,s^*}$ and such that honest nodes trigger the output of $\Pi_{\text{ab}}.\mathsf{Output}(s^*, (B, \sigma))$, where B is a valid block (Alg. 4, lns. 9, 11 and 13) that indicates the batch of proposer p as available (Alg. 4, ln. 20). Let t_0 be the earliest time any honest node triggers the output of $\Pi_{\text{ab}}.\mathsf{Output}(s^*, (B, \sigma))$. Following the arguments for safety, selective-censorship resistance, and liveness, once Π_{ab} confirms B, tx_b position in L^*_∞ of $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ is implicitly confirmed: under the given assumptions, reconstruction of slot s^* is "deterministic" in that eventually, slot $s^* - 1$ will complete reconstruction, B deems the batch of proposer p as available, sufficiently many relays will release their shreds for $\boldsymbol{m}_{p,s^*}$, all honest nodes reconstruct $\boldsymbol{m}_{p,s^*}$, and finally all honest nodes include the transactions from $\boldsymbol{m}_{p,s^*}$ into their logs for s^*. It follows that tx_b is in the valency log of $\Pi_{\text{mcp}}(\Pi_{\text{ab}})$ by time t_0, so $t_0 \geq t^*$.

The earliest time any honest node broadcasts a shred for a batch $\boldsymbol{m}_{p,s}$ containing tx_b, is after Π_{ab} confirms a block attesting to $\boldsymbol{m}_{p,s}$ as available, which is no earlier than t_0. Thus, since $t_0 \geq t^*$, until t^*, the only information honest relays expose about the batches p submits, are signatures on the commitments given to them by p (Alg. 3, ln. 27). Since these commitments are already known to the adversary, these signatures reveal no information about the batches, and we can ignore them going forward. Crucially, the adversary does not learn any information about $\boldsymbol{m}_{p,s}$ from honest shreds until t_0, which is after t^*.

Let $\mathcal{S}$ be the set of slots that elapse between t^* and the time tx_b is submitted to p. Note that while the adversary has access to all the information stretching back to the first slot, no slots before the time tx_b is submitted, contain information to help the adversary determine b.

Hence, up to time t^*, for each slot s, the only information the adversary learns about $\boldsymbol{m}_{p,s}$ is from the information (commitments and openings) submitted to Byzantine relays (Alg. 3, ln. 19). Namely, let Q_s denote the set of adversary controlled relays for slot s. Then, the adversary's view with respect to $\boldsymbol{m}_{p,s}$ consists of $V_s = \{(C, c_{\mathcal{N}_{\text{relay}}(s)^{-1}[r]}, r_{\mathcal{N}_{\text{relay}}(s)^{-1}[r]}, w_r, \sigma_r) \mid r \in Q_s\}$ (Alg. 3, ln. 19). Since $|Q_s| \leq f_{\text{relay}}$, and $\mathsf{HECC.Enc}$ is instantiated with $T = \tau n_{\text{relay}} \geq f_{\text{relay}}$, this is exactly the setting of Lem. 4. Thus, by Lem. 4, we have that from V_s, the adversary cannot discern whether tx_0 or tx_1 is in $\boldsymbol{m}_{p,s}$.

For every slot, p generates fresh randomness to use in $\mathsf{HECC.Enc}$ (Alg. 3, ln. 11) and $\mathsf{VC.Commit}$ (Alg. 3, ln. 13). As a result, from the relevant part of the adversary's view, $V_{\mathcal{S}} = \{V_s \mid s \in \mathcal{S}\}$, the adversary cannot distinguish if the batches $\{\boldsymbol{m}_{p,s}\}_{s \in \mathcal{S}}$ contain tx_0 or tx_1. □

5 Discussion

5.1 Tradeoffs in Probabilistic Guarantees

The MCP protocol of Sec. 3 is always safe. But, liveness, selective-censorship resistance, and hiding are only satisfied with overwhelming probability. In the parameterization analyzed in Sec. 4, all three properties only hold when the adversary controls fewer than $n_{\text{relay}}/5$ relays per slot. Thus, if n_{relay} is set low, an adversary might get lucky and have an outsized number of its nodes sampled as relays for a given slot, and then threaten these properties of the protocol.

However, a protocol might favor some of these properties over others. A natural choice is for the protocol to preserve liveness against a stronger adversary, to render the probability of a liveness fault for moderately-strong adversaries extremely low, at the cost of a weaker adversary being able to break selective-censorship resistance or hiding occasionally.

As described in Sec. 4, a liveness fault requires $f_{\text{relay}} > (\varphi - \gamma)n_{\text{relay}}$, selective-censorship resistance can be violated if $f_{\text{relay}} > (\mu - \varphi)n_{\text{relay}}$, and hiding can be violated when $f_{\text{relay}} > T$. Thus, we can bias MCP towards favoring liveness by increasing the availability threshold φ, and/or decreasing T (corresponding to a lower γ). We give one such choice of these parameters tuned to favor liveness, along with the probabilities of each of the properties being violated for a fixed slot.

The number of Byzantine relays sampled in a given slot follows a binomial distribution, $X \sim \text{Binomial}(n_{\text{relay}}, f)$. We can then compute, as a function of $n_{\text{relay}}, f, \gamma, \mu, \varphi, T$, the probability of a potential liveness fault, selective-censorship fault, and hiding fault in a given slot. These probabilities are $\mathbb{P}[X > (\varphi - \gamma)n_{\text{relay}}]$, $\mathbb{P}[X > (\mu - \varphi)n_{\text{relay}}]$, and $\mathbb{P}[X > T]$, respectively.

For instance, setting $n_{\text{relay}} = 512$, $\gamma = 0.3$, $T = n_{\text{relay}} \cdot 0.15$, $K = n_{\text{relay}} \cdot 0.15$, $\varphi = 0.55$, $\mu = 0.8$, and $f = 0.15 \cdot n_{\text{relay}}$, we get that both the probability of a potential liveness failure and the probability of a potential selective-censorship failure is roughly 1 in 10^9 per slot, which is once every 10 years given 400 millisecond slot times. In this case, since $f = T$, the probability of hiding being broken in every slot is fairly high. But, we reiterate that T shreds is only the threshold for which it is information-theoretically possible for an adversary to learn *some* information about the batch. In practice, an adversary would likely need more shreds before they can glean *useful* information about batch contents. Furthermore, if a proposer desires to hide its transactions against stronger adversaries, it can unilaterally decide to dedicate part of its batch to additional

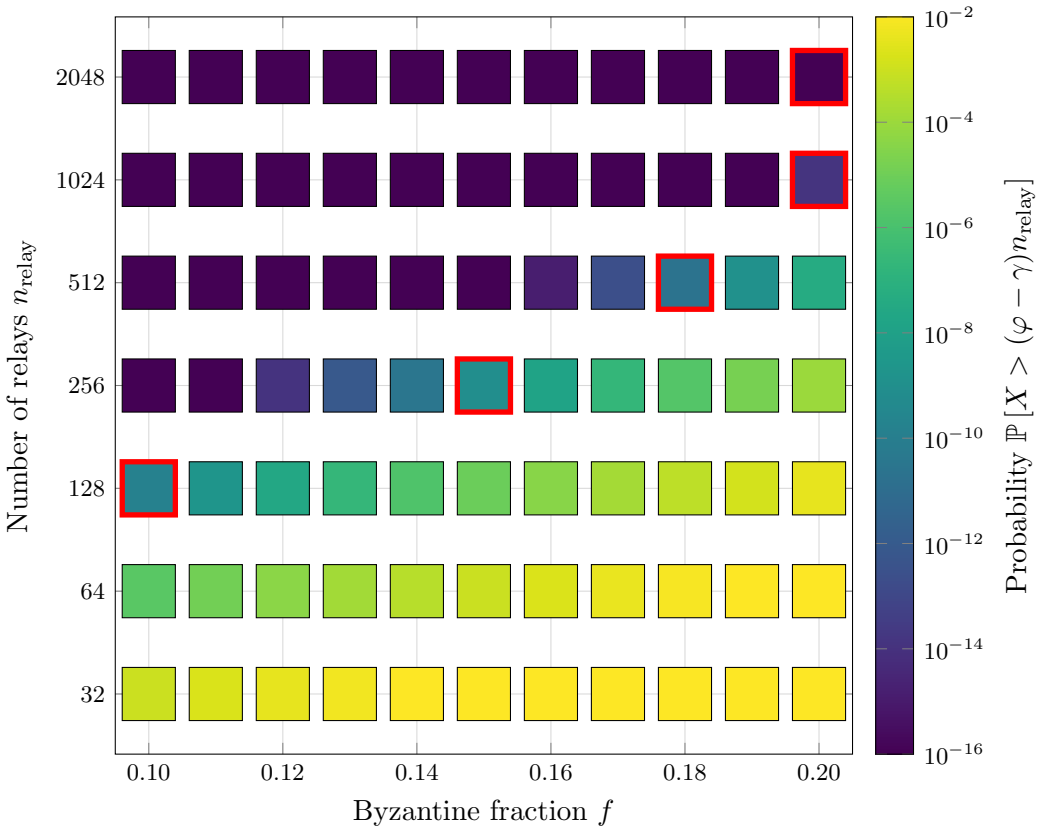

Figure 3: Probability $\mathbb{P}[X > (\varphi - \gamma)n_{\text{relay}}]$ under different parameter settings for the MCP protocol. The heatmap shows how the probability $\mathbb{P}[X > (\varphi - \gamma)n_{\text{relay}}]$ varies with Byzantine fraction f and number of relays n_{relay}, for $\varphi = 0.6$ and $\gamma = 0.3$. Red rectangles highlight the highest f for a given n_{relay} at which a probability of at most $1 : 10^9$ is maintained.

randomness. Given that the regular operating mode of the protocol is with parameters K and T for encoding batches, a proposer can instead encode their batch with parameters K', T' such that $T' \geq T$ and $K' + T' = K + T$, but a proposer will always be capped by γ on the maximum resilience they can get for hiding.

5.2 Recovering Liveness

Even in cases where honest nodes are left unable to reconstruct batches deemed available, due to missing/withheld shreds blocking the confirmation of subsequent payloads, the underlying atomic broadcast protocol Π_{ab} still makes progress independently (Alg. 3 does not wait for any response from reconstruction before making a new block). We can use this to construct a liveness recovery gadget, where nodes reach consensus on the "unavailability of a batch". For instance, if nodes are blocked on receiving enough shreds for a given batch, they can submit a vote into Π_{ab} indicating a specific batch as being unavailable. Once a quorum of such votes is gathered, a "batch-skip certificate" is produced, certifying it is safe for nodes to skip reconstruction. Thus, either some

honest node will eventually gather enough shreds to reconstruct a batch and forward the contents to all other nodes, or nodes will eventually agree to skip the batch.[8]

From here, there are multiple options for how to resume batch reconstruction/confirmation, including: (a) Full rollback. All blocks proposed from the failed slot up to the issuance of the skip certificate are not confirmed. This approach is simple, but may discard batches that could be successfully reconstructed. (b) Isolated skip. Only the failed slot is skipped, while subsequent slots that can be reconstructed are preserved. This could be further refined to operate per-batch rather than per-slot. A challenge with this approach is the bursty bandwidth required to release a backlog of batches deemed available but confirmed only at a later time.

5.3 Choosing Proposers and Allocating Blockspace

A degree of freedom in implementing MCP is the selection of proposers and the allocation of blockspace among them. Since execution and networking resources are bounded, individual batches must be capped in size to ensure feasibility of reconstruction and timely confirmation.

An advantage of the MCP construction is that the communication overhead scales with the aggregate blocksize, rather than with the number of proposers. Specifically, if a batch u is partitioned into M chunks $u_1, ..., u_M$, each individually encoded, then a single proposer transmitting u produces the same total number of shreds as k proposers jointly transmitting disjoint sub-batches whose concatenation equals u. Such partitioning is required regardless of the number of proposers, as all transactions for a block are unlikely to fit inside one n_{relay}-length codeword. Thus, the incremental overhead of additional proposers consists only of one extra commitment per proposer, and the associated signature verifications by relays and the leader. This cost is modest and permits a large number of concurrent proposers.

A natural proposer selection rule is stake-weighted eligibility: every node holding at least a minimum stake fraction (e.g., 0.2%) is eligible as a proposer, with blockspace proportional to its stake. Nodes unwilling to propose themselves, may delegate their blockspace to other participants. Consequently, the proposer set remains relatively stable, while the barrier to participation remains low enough that clients can reliably route transactions to at least one honest proposer.

5.4 Localized Relay Optimization

The protocol described in Sec. 3 takes 5 rounds. One round can be saved through a localized relay optimization. Each validator would maintain relays in several locations around the world. After Alg. 3, ln. 29, the relay privately forwards its shreds to its sibling (relays from the same validator). Each of the local relays listens for the output of Π_{ab}, and broadcasts shreds as soon as the broadcast conditions are met (Alg. 4, ln. 24).

[8] As stated here, such a gadget would make the MCP protocol unsafe in the case where an adversary releases shreds so that some nodes successfully reconstruct a batch (and hence confirm the corresponding transactions) while other nodes vote to skip the batch. This can be addressed, at the cost of additional communication/latency, by having nodes submit additional votes certifying successful reconstruction before confirming batches.

Acknowledgment

We thank Ittai Abraham, Jeff Bezaire, Dan Boneh, Joseph Bonneau, Yuval Efron, Quentin Kniep, Maher Latif, Kartik Nayak, Mallesh Pai, Tom Pointon, Guru-Vamsi Policharla, Ling Ren, Dan Robinson, Tim Roughgarden, Jakub Sliwinski, Alberto Sonnino, Ertem Nusret Tas, and Roger Wattenhofer for fruitful discussions. Special thanks to Yuval Efron with whom we have discussed extensions of valency to multi-shot consensus (like Def. 10) on various occasions.

References

[1] Interleaved RS (IRS) code, 2022. The Error Correction Zoo, accessed 2025-09-07. URL: `https://errorcorrectionzoo.org/c/interleaved_reed_solomon`.

[2] Amit Agarwal, Rex Fernando, and Benny Pinkas. Efficiently-thresholdizable batched identity based encryption, with applications. In *CRYPTO (3)*, volume 16002 of *Lecture Notes in Computer Science*, pages 69–100. Springer, 2025.

[3] Orestis Alpos, Bernardo David, Nikolas Kamarinakis, and Dionysis Zindros. An overview of the efficiency and censorship-resistance guarantees of widely-used consensus protocols, 2025. `arXiv:2504.03588v2`.

[4] Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. Shoal++: High throughput DAG BFT can be fast and robust! In *NSDI*, pages 813–826. USENIX Association, 2025.

[5] Hagit Attiya and Faith Ellen. *Impossibility Results for Distributed Computing*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2014.

[6] Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. Mysticeti: Reaching the latency limits with uncertified dags. In *NDSS*. The Internet Society, 2025.

[7] Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 590–609. Springer, 2011.

[8] Maryam Bahrani, Pranav Garimidi, and Tim Roughgarden. Centralization in block-building and proposer-builder separation. In *FC (1)*, volume 14744 of *Lecture Notes in Computer Science*, pages 331–349. Springer, 2024.

[9] Soumya Basu, Alin Tomescu, Ittai Abraham, Dahlia Malkhi, Michael K. Reiter, and Emin Gün Sirer. Efficient verifiable secret sharing with share recovery in BFT protocols. In *CCS*, pages 2387–2402. ACM, 2019.

[10] Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive, Paper 2022/898, 2022. URL: `https://eprint.iacr.org/2022/898`.

[11] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In *TCC (1)*, volume 12550 of *Lecture Notes in Computer Science*, pages 260–290. Springer, 2020.

[12] G. R. Blakley. Safeguarding cryptographic keys. In *MARK*, pages 313–318. IEEE, 1979.

[13] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO (1)*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

[14] Dan Boneh, Benedikt Bünz, Kartik Nayak, Lior Rotem, and Victor Shoup. Context-dependent threshold decryption and its applications. Cryptology ePrint Archive, Paper 2025/279, 2025. URL: `https://eprint.iacr.org/2025/279`.

[15] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003.

[16] Dan Boneh, Evan Laufer, and Ertem Nusret Tas. Batch decryption without epochs and its application to encrypted mempools. Cryptology ePrint Archive, Paper 2025/1254, 2025. URL: `https://eprint.iacr.org/2025/1254`.

[17] Jan Bormet, Arka Rai Choudhuri, Sebastian Faust, Sanjam Garg, Hussien Othman, Guru-Vamsi Policharla, Ziyan Qu, and Mingyuan Wang. BEAST-MEV: Batched threshold encryption with silent setup for MEV prevention. Cryptology ePrint Archive, Paper 2025/1419, 2025. URL: `https://eprint.iacr.org/2025/1419`.

[18] Jan Bormet, Sebastian Faust, Hussien Othman, and Ziyan Qu. BEAT-MEV: Epochless approach to batched threshold encryption for MEV prevention. Cryptology ePrint Archive, Paper 2024/1533, 2024. URL: `https://eprint.iacr.org/2024/1533`.

[19] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, Sergey Nazarov, Alexandru Topliceanu, Florian Tramèr, and Fan Zhang. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks, 2021. Chainlink 2.0 whitepaper v1.0, accessed 2025-09-07. URL: `https://research.chain.link/whitepaper-v2.pdf`.

[20] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus, 2018. `arXiv:1807.04938v3`.

[21] Eric Budish, Peter Cramton, and John Shim. The high-frequency trading arms race: Frequent batch auctions as a market design response. *Quarterly Journal of Economics*, 130(4):1547–1621, 2015. `doi:10.1093/qje/qjv027`.

[22] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *CCS*, pages 88–97. ACM, 2002.

[23] Christian Cachin and Jovana Micic. Quick order fairness: Implementation and evaluation. In *ICBC*, pages 230–234. IEEE, 2024.

[24] Christian Cachin, Jovana Micic, Nathalie Steinhauer, and Luca Zanolini. Quick order fairness. In *Financial Cryptography*, volume 13411 of *Lecture Notes in Computer Science*, pages 316–333. Springer, 2022.

[25] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[26] Benjamin Y. Chan and Rafael Pass. Simplex consensus: A simple and fast consensus protocol. In *TCC (4)*, volume 14372 of *Lecture Notes in Computer Science*, pages 452–479. Springer, 2023.

[27] Benjamin Y. Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains. In *AFT*, pages 1–11. ACM, 2020.

[28] James Hsin-yu Chiang, Bernardo David, Ittay Eyal, and Tiantian Gong. Fairpos: Input fairness in permissionless consensus. In *AFT*, volume 282 of *LIPIcs*, pages 10:1–10:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[29] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *FOCS*, pages 383–395. IEEE Computer Society, 1985.

[30] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In *USENIX Security Symposium*. USENIX Association, 2024.

[31] Michele Ciampi, Aggelos Kiayias, and Yu Shen. Universal composable transaction serialization with order fairness. In *CRYPTO (2)*, volume 14921 of *Lecture Notes in Computer Science*, pages 147–180. Springer, 2024.

[32] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *SP*, pages 910–927. IEEE, 2020.

[33] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *EuroSys*, pages 34–50. ACM, 2022.

[34] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *CCS*, pages 2705–2721. ACM, 2021.

[35] Sourav Das, Zhuolun Xiang, Alin Tomescu, Alexander Spiegelman, Benny Pinkas, and Ling Ren. Verifiable secret sharing simplified. In *SP*, pages 633–651. IEEE, 2025.

[36] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. Mcfly: Verifiable encryption to the future made practical. In *FC (1)*, volume 13950 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2023.

[37] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[38] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, pages 427–437. IEEE Computer Society, 1987.

[39] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[40] Elijah Fox, Mallesh M. Pai, and Max Resnick. Censorship resistance in on-chain auctions. In *AFT*, volume 282 of *LIPIcs*, pages 19:1–19:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[41] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *STOC*, pages 699–710. ACM, 1992.

[42] Lioba Heimbach, Lucianna Kiffer, Christof Ferreira Torres, and Roger Wattenhofer. Ethereum's proposer-builder separation: Promises and realities. In *IMC*, pages 406–420. ACM, 2023.

[43] Lioba Heimbach and Roger Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. In *AFT*, pages 47–60. ACM, 2022.

[44] Nicholas A. G. Johnson, Theo Diamandis, Alex Evans, Henry de Valence, and Guillermo Angeris. Concave pro-rata games. In *FC Workshops*, volume 13953 of *Lecture Notes in Computer Science*, pages 266–285. Springer, 2023.

[45] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[46] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Revised Third Edition*. CRC Press, 2025.

[47] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. In *CCS*, pages 475–489. ACM, 2023.

[48] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *CRYPTO (3)*, volume 12172 of *Lecture Notes in Computer Science*, pages 451–480. Springer, 2020.

[49] Quentin Kniep, Kobi Sliwinski, and Roger Wattenhofer. Solana Alpenglow consensus, increased bandwidth, reduced latency, 2025. Alpenglow whitepaper v1.1 (July 22, 2025), Anza blog post, accessed 2025-09-07. URL: `https://www.anza.xyz/blog/alpenglow-a-new-consensus-for-solana`.

[50] Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *AFT*, pages 25–36. ACM, 2020.

[51] Andrew Lewis-Pye and Tim Roughgarden. Permissionless consensus, 2023. `arXiv:2304.14701v5`.

[52] Dahlia Malkhi and Pawel Szalachowski. Maximal extractable value (MEV) protection on a DAG. In *Tokenomics*, volume 110 of *OASIcs*, pages 6:1–6:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[53] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In *CCS*, pages 31–42. ACM, 2016.

[54] Ciamac C. Moallemi, Mallesh M. Pai, and Dan Robinson. Latency advantages in common-value auctions, 2025. `arXiv:2504.02077v1`.

[55] Peyman Momeni, Sergey Gorbunov, and Bohan Zhang. Fairblock: Preventing blockchain front-running with minimal overheads. In *SecureComm*, volume 462 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 250–271. Springer, 2022.

[56] Ke Mu, Bo Yin, Alia Asheralieva, and Xuetao Wei. Separation is good: A faster order-fairness byzantine consensus. In *NDSS*. The Internet Society, 2024.

[57] Kamilla Nazirkhanova, Joachim Neu, and David Tse. Information dispersal with provable retrievability for rollups. In *AFT*, pages 180–197. ACM, 2022.

[58] Alex Obadia. Flashbots: Frontrunning the MEV crisis, 2020. Flashbots Medium blog post, accessed 2025-09-07. URL: `https://medium.com/flashbots/frontrunning-the-mev-crisis-40629a613752`.

[59] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer, 1991.

[60] Geoffrey Ramseyer, Ashish Goel, and David Mazières. SPEEDEX: A scalable, parallelizable, and economically efficient decentralized exchange. In *NSDI*, pages 849–875. USENIX Association, 2023.

[61] Geoffrey Ramseyer and David Mazières. Groundhog: Linearly-scalable smart contracting via commutative transaction semantics, 2024. `arXiv:2404.03201v1`.

[62] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of The Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960. `doi:10.1137/0108018`.

[63] Max Resnick. Execution consensus separation, 2024. Ethresear.ch forum post #19964, accessed 2025-09-07. URL: `https://ethresear.ch/t/execution-consensus-separation/19964`.

[64] Dan Robinson and David White. Priority is all you need, 2024. Paradigm blog post, accessed 2025-09-07. URL: `https://www.paradigm.xyz/2024/06/priority-is-all-you-need`.

[65] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[66] Victor Shoup and Nigel P. Smart. Lightweight asynchronous verifiable secret sharing with optimal resilience. *J. Cryptol.*, 37(3):27, 2024.

[67] Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, and Kartik Nayak. Sailfish: Towards improving the latency of dag-based BFT. In *SP*, pages 1928–1946. IEEE, 2025.

[68] Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. Shoal: Improving DAG-BFT latency and robustness. In *FC (1)*, volume 14744 of *Lecture Notes in Computer Science*, pages 92–109. Springer, 2024.

[69] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In *CCS*, pages 2705–2718. ACM, 2022.

[70] Nirvan Tyagi, Arasu Arun, Cody Freitag, Riad S. Wahby, Joseph Bonneau, and David Mazières. Riggs: Decentralized sealed-bid auctions. In *CCS*, pages 1227–1241. ACM, 2023.

[71] Apostolos Tzinas, Srivatsan Sridhar, and Dionysis Zindros. On-chain timestamps are accurate. In *FC (1)*, volume 14744 of *Lecture Notes in Computer Science*, pages 110–127. Springer, 2024.

[72] Uniswap Labs and Flashbots. Live on Unichain: Fair transaction ordering and MEV protection, 2025. Uniswap Labs blog post, accessed 2025-09-07. URL: `https://blog.uniswap.org/rollup-boost-is-live-on-unichain`.

[73] Mohammad Amin Vafadar and Majid Khabbazian. Condorcet attack against fair transaction ordering. In *AFT*, volume 282 of *LIPIcs*, pages 15:1–15:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[74] Dave White, Dan Robinson, Ludwig Thouvenin, and Karthik Srinivasan. Leaderless auctions, 2024. Paradigm blog post, accessed 2025-09-07. URL: `https://www.paradigm.xyz/2024/02/leaderless-auctions`.

[75] Jarry Xiao, @FrankieIsLost, 0xShitTrader, and Dan Robinson. A sandwich-resistant AMM, 2024. Accessed 2025-09-07. URL: `https://www.umbraresearch.xyz/writings/sandwich-resistant-amm`.

[76] Bowen Xue, Soubhik Deb, and Sreeram Kannan. Bigdipper: A hyperscale bft system with short term censorship resistance, 2023. `arXiv:2307.10185v3`.

[77] Bowen Xue and Sreeram Kannan. Travelers: A scalable fair ordering bft system, 2024. `arXiv:2401.02030v1`.

[78] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. DispersedLedger: High-throughput byzantine consensus on variable bandwidth networks. In *NSDI*, pages 493–512. USENIX Association, 2022.

[79] Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. Sok: Mev countermeasures: Theory and practice, 2022. `arXiv:2212.05111v2`.

[80] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356. ACM, 2019.

[81] Haoqian Zhang, Louis-Henri Merino, Ziyan Qu, Mahsa Bastankhah, Vero Estrada-Gali textasciitilde nanes, and Bryan Ford. F3B: A low-overhead blockchain architecture with per-transaction front-running protection. In *AFT*, volume 282 of *LIPIcs*, pages 3:1–3:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[82] Yunhao Zhang, Srinath T. V. Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without byzantine oligarchy. In *OSDI*, pages 633–649. USENIX Association, 2020.

Contents

1 Introduction **1**
- 1.1 Three Examples . . . 2
- 1.2 Related Work . . . 4

2 Preliminaries & Model **7**
- 2.1 Cryptographic Primitives . . . 7
  - 2.1.1 Signatures . . . 7
  - 2.1.2 Vector Commitments . . . 8
  - 2.1.3 Hiding Erasure-Correcting Codes . . . 13
  - 2.1.4 Combining Vector Commitments and Hiding Erasure-Correcting Codes . . . 14
- 2.2 Setting, Adversary, Network . . . 16
- 2.3 Consensus Basics . . . 17
  - 2.3.1 Safety and Liveness of Atomic Broadcast . . . 17
  - 2.3.2 Slightly Enriched Interface . . . 17
- 2.4 Economic Consensus Properties . . . 18
  - 2.4.1 Selective-Censorship Resistance . . . 18
  - 2.4.2 Hiding . . . 18
  - 2.4.3 Practical Considerations of Selective-Censorship Resistance and Hiding . . . 19

3 Multiple Concurrent Proposers Protocol **20**
- 3.1 Philosophy & Overview . . . 20
- 3.2 Detailed Description . . . 22
  - 3.2.1 Setup (Alg. 3, ln. 1) . . . 22
  - 3.2.2 Proposer Phase (Alg. 3, ln. 8) . . . 22
  - 3.2.3 Relay Phase (Alg. 3, ln. 20) . . . 25
  - 3.2.4 Consensus Phase (Alg. 3, ln. 30) . . . 25
  - 3.2.5 Reconstruction Phase (Alg. 4, ln. 7) . . . 25

4 Analysis **26**
- 4.1 Parameter Choices . . . 26
- 4.2 Safety . . . 27
- 4.3 Selective-Censorship Resistance . . . 29
- 4.4 Liveness . . . 30
- 4.5 Hiding . . . 31

5 Discussion **32**
- 5.1 Tradeoffs in Probabilistic Guarantees . . . 32
- 5.2 Recovering Liveness . . . 33
- 5.3 Choosing Proposers and Allocating Blockspace . . . 34
- 5.4 Localized Relay Optimization . . . 34