The Espresso Sequencing Network: HotShot Consensus, Tiramisu Data-Availability, and Builder-Exchange

Jeb Bearer[1], Benedikt Bünz[1,3], Philippe Camacho[1], Binyi Chen[1,4], Ellie Davidson[1], Ben Fisch[1,2], Brendon Fish[1], Gus Gutoski[1], Fernando Krell[1], Chengyu Lin[1], Dahlia Malkhi[1,6,7], Kartik Nayak[1,5], Keyao Shen[1], Alex Xiong[1], Nathan Yospe[1]

[1]Espresso Systems, [2]Yale University, [3]New York University, [4]Stanford University, [5]Duke University, [6]Chainlink Labs, [7]UC Santa Barbara
{firstname}@espressosys.com

Abstract

Building a Consensus platform for shared sequencing can power an ecosystem of layer-2 solutions such as rollups which are crucial for scaling blockchains (e.g.,Ethereum). However, it drastically differs from conventional Consensus for blockchains in two key considerations:

- (No) Execution: A shared sequencing platform is not responsible for pre-validating blocks nor for processing state updates. Therefore, agreement is formed on a sequence of certificates of block data-availability (DA) without persisting them or obtaining blocks in full. At the same time, the platform must stream block data with very high efficiency to layer-2 entities for execution, or (in the case of rollups) for proof generation.
- Builder-Exchange: A shared sequencing platform delegates to external entities to build blocks and separates it from the role of a consensus proposer. This allows an ecosystem of specialized builders to pre-validate transactions for diversified rollups, languages, and MEV exploits. However, separating the task of block-building from proposing brings a new challenge. Builders want assurances that their blocks would commit in exchange for revealing their contents, whereas validators/proposers want assurance that the data in committed blocks will be available and fees paid. Neither one trusts the other, hence the shared sequencing platform should facilitate a "fair-exchange" between builders and the sequencing network.

The Espresso Sequencing Network is purpose-built to address these unique considerations. **Among the main novelties of the design are (i) a three-layered DA system called Tiramisu, coupled with (ii) a costless integration of the DA with the platform's consensus core, and (iii) a Builder-Exchange mechanism between builders and the consensus core.**

Note that this paper relies substantially on and can be seen as an extension of *The Espresso Sequencer: HotShot Consensus and Tiramisu Data Availability* [84].

Contents

1 Introduction

Building a Consensus platform for *shared sequencing* can power an ecosystem of *layer-2* solutions such as rollups [51, 89, 83] which are crucial for scaling blockchains (e.g., Ethereum). However, it drastically differs from conventional Consensus for blockchains in two key considerations:

(No) Execution: A shared sequencing platform is not responsible for pre-validating blocks nor for processing state updates. Therefore, the validator nodes of the sequencing network do not need to obtain, nor persist, full copies of blocks. Rather, they form agreement on a sequence of certificates of block *data-availability* (DA). At the same time, the platform must stream block data with very high efficiency to layer-2 entities for execution, or (in the case of rollups) for proof generation.

Builder-Exchange: A shared sequencing platform delegates to external entities to *build blocks* and separates it from the role of a consensus proposer. This allows an ecosystem of specialized builders to pre-validate transactions for diversified rollups, languages, and MEV exploits. However, separating the task of block-building from proposing brings a new challenge.

Builders want assurances that their blocks would commit in exchange for revealing their contents, whereas validators/proposers want assurance that the data in committed blocks will be available and fees paid. Neither one trusts the other, hence the shared sequencing platform should facilitate a "fair-exchange" between builders and the sequencing network.

The Espresso Sequencing Network is purpose-built to address these unique considerations. **Among the main novelties of the design are (i) a three-layered DA system called Tiramisu, coupled with (ii) a costless integration of the DA with the platform's consensus core, and (iii) a Builder-Exchange mechanism between builders and the consensus core.**

The need for a shared decentralized sequencing network. Layer-2 (or L2) solutions such as rollups [51, 89, 83] have been introduced to improve the transaction processing performance on top of Ethereum. By handling execution outside the Layer-1, rollups have become the de facto solution for scaling Ethereum [17]. The immediate benefits for users are obvious: Low latency, high throughput, fast confirmations and above all low gas fees. Misfortunately the price to pay for this convenience is high: First of all L2 sequencers are in practice centralized which expose users to liveness issues [32] and the risk of being censored. While decentralizing the rollup sequencer could be a solution, achieving the same economic security as Ethereum is likely to be challenging. Moreover, another fundamental problem remains: By default L2s do not interoperate in a smooth way. This means we now have to deal with fragmented user bases and liquidity in addition to encouraging economic centralization due to cross-chain MEV opportunities that are only accessible to well funded players [65, 66].

A shared sequencing network can support atomic execution of a set of transactions belonging to multiple rollups, allowing users to trade cross-rollup without risk, opening new possibilities such as cross-rollup flash-loans and others. Indeed, assuming the shared sequencing infrastructure is in place, recent proposals such as AggLayer [78] suggest that trustless synchronous interactions between L2s are possible.

HotShot design principles. The motivation above leads us to introduce HotShot, a decentralized, highly performant network for shared sequencing. In designing HotShot, several design principles are combined.

First, scaling in the number of validator nodes is crucial for achieving strong security through decentralization. This level of decentralization is achieved by adopting a proof-of-stake participation regime.

Second, *bribery* attacks, where an adversary, even without knowing who to corrupt, can advertise payouts for certain verifiable malicious behaviors [9],(e.g., the attacker can create a smart contract that pays participants to censor specific transactions) are a concern. An adaptive, bribing adversary is one of the strongest threat models, and is discussed further in [84]. To achieve bribery resistance, all staked nodes secure the safety of the protocol, rather than relying on any kind of sub-committee solution. This security level is enabled by utilizing a consensus core with **linear complexity** in the optimistic case, which is based on HotStuff-2 [63] coupled with the view synchronization protocol from Naor and Keidar [70].

Third, optimizing for quick response in optimistic conditions is a priority. The steady state protocol of HotShot achieves an important property called *optimistic responsiveness*, advancing at the speed of the underlying transport when network conditions are favorable.

The HotShot consensus core is described in Section 5. We proceed to describe the DA layer, the Builder-Exchange mechanism, and their integration with HotShot.

Tiramisu: The Three-Layered Data Availability Solution Our data availability solution, Tiramisu, is designed to balance two requirements. On one hand, we want certifiable data dissemination without fully replicating information and without introducing extra steps to the consensus protocol. On the other hand, we want efficient retrieval that (in the common case) doesn't need to collect and process pieces of data. These requirements are met through three mechanisms, mirroring the layers of the timeless Italian dessert.

At the base, the *Savoiardi* layer, uses a newly-introduced variant of the verifiable information dispersal (VID) scheme from [7] to guarantee data availability. The idea is to encode the data block into erasure-coded chunks and send one chunk per node. Nodes that receive valid chunks return a signed acknowledgement, of which a quorum certifies the availability of data against corruption of less than one third of the nodes. (As we shall see below, the Savoiardi certificate of availability double-serves in consensus as a quorum of *votes*.) To destroy data availability, an adversary would need to control 1/3-fraction of the stake in the system.

A disadvantage of Savoiardi is the cost of retrieval. A rollups who wishes to recover the full payload must download shares from many storage nodes and spend computation resources to decode the payload. To remedy this, we introduce the *Mascarpone* layer. This layer enables fast reconstruction through a randomly elected small random committee. Each node in this committee receives the entire block data. A valid DA certificate must include a signature by a threshold, e.g. 80%, of the committee. This ensures that with high probability, every quorum of two-thirds will include one honest node that can provide fast access to the data and aid reconstruction. The random selection ensures that the committee can be small, but unfortunately, this also makes it more vulnerable to briberies. The Savoiardi layer does not have this vulnerability and provides the strongest security. Combining Savoiardi and Mascarpone gives both strong security and fast reconstruction.

Finally, we can optimistically increase the performance of the system by adding a content delivery network (CDN), which we call the Cocoa layer. The CDN can cache and efficiently distribute data, and can be thought of as an efficient broadcast layer. It can deliver performance on the level of traditional Web2 infrastructure, but it is entirely optional and is backed up by the Mascarpone and Savoiardi layers. When the CDN is online, it can significantly improve the performance of those layers and the system overall, by quickly disseminating data and providing fast access to it. The Cocoa layer aligns well with our *optimistically responsive* goal, where the solution works faster when all components are fully working and online, but still gives high-security guarantees in the presence of an adversary.

In terms of complexity, the Savoiardi VID scheme has linear message complexity for storing/retrieving data, albeit only constant bit costs. The Mascarpone and Cocoa layers have constant message complexity for storing/retrieving. Retrieval from Mascarpone or Cocoa is fast and has a constant communication cost. A bribing adversary might corrupt the entire random DA committee, forcing the protocol to use Savoiardi to retrieve the data. This does not hurt liveness but can cause a slowdown and inefficiency. However, a cheaper attack would slow down consensus leaders anyway. Thus, we expect that Savoiardi will seldom be triggered for data retrieval, and the data can be retrieved quickly from CDN nodes or the small DA committee in the optimistic condition where the leader is honest and the DA committee is not bribed.

Integration of Tiramisu DA with HotShot Consensus. HotShot is a view-by-view protocol based on HotStuff-2: each view has a leader that proposes a block (possibly outsourcing the construction of the block to a third party) to extend the sequence of blocks. In order to drive a consensus decision, the leader must collect a quorum of votes forming a Quorum Certificate ("QC") on its proposal. For liveness, the nodes in HotShot consensus need to ensure data availability before voting for a vector commitment proposal. This is done by stipulating that a node votes on a block proposal only if (i) the proposal carries a certificate that a threshold of parties in the random DA committee received the full data, and (ii) the node itself received a Savoiardi piece for the commitment. In this way, when a QC is formed towards a sequencing decision, it a fortiori guarantees that a block's data will always be available.

Integration of Tiramisu DA with a Builder-Exchange. Separating the task of block-building from consensus validation, so that it is performed by two different entities, brings inherent challenges that relate to the trust between the two:

Builders concern: Builders are concerned about revealing blocks before they are committed to slots in the blockchain. The predominant reason is that block builders in ecosystems like Ethereum invest substantial effort in searching and exploiting MEV opportunities [8], and they do not want validators to steal [30] what they discovered. Additionally, builders may be willing to pre-pay a fee to validators in the form of a bid; however, if blocks are not included in the designated slots, they would lose the fee.

Validators concern: Validators are concerned about committing blocks without having access to the contents because if the data is not available, they may lose their block rewards and fees. It also leaves the blockchain with "holes".

Currently, the concrete solution available for Ethereum to the fair-exchange problem consists of relying on trusted third parties called *relays* [44]. Misfortunately, these new single-point-of-failures have already yielded exploits [31] and are not economically sustainable in the long run [11]. While decentralized solutions to this problem have been proposed [18], their practicality is still a matter of debate [72].

The Espresso Sequencing Network addresses this dilemma via a Builder-Exchange mechanism between builders and the HotShot consensus core. The trick is a Builder-driven approach that delegates the privilege to propose blocks to the builder which wins a slot (and pays a fee). The winning builder further uses a commit-reveal scheme that hides the block contents until after the builder itself is assured that it can drive a commit decision. This exchange hinges on a unique capability of the HotShot platform: the builder sends a fee payment which is **predicated on a consensus decision** that includes its block at the designated slot. HotShot uniquely supports such predicated payments.

The Builder-Exchange mechanism deals with all but the least probable threats. Briefly,once the builder reveals enough pieces of the block, the builder itself obtains a QC. Even a bribing adversary cannot bribe a small subset (e.g., the next proposer or a DA committee) to reveal the block content in advance of proving the builder with a QC. Once obtaining a QC, a bad proposer or a small subset of validators cannot hold back the builder from driving its block to commit. Even if a majority of validators prevent the commit, they will simply be lose progress and be left without the fee. Finally, while the builder could send a bogus decryption key, this is not a concern, validators form agreement on what the (bogus) block is.

2 Architecture Overview

In this section, we briefly describe the main components of the Espresso Sequencing Network, as well as their interactions with users, builders, L2 rollups and the L1 blockchain (e.g., Ethereum).

The Espresso Sequencing Network is a decentralized network of thousands of heterogeneous nodes providing *log replication* as a service. Internally, it consists of three core components:

1. The *HotShot Consensus layer* (Section 5) that enables builders to sequence transactions without handling execution. It is a permissionless, leader-based, multi-shot BFT protocol that extends HotStuff-2 [63] to the proof-of-stake setting.
2. The *Tiramisu DA layer* (Section 3) that guarantees the availability of the data submitted by builders. More specifically, a set of storage nodes are responsible for the availability and retrievability of raw transaction data. Since the Espresso Sequencing Network is not responsible for execution, the data need not be broadcast to every node; this is essential for achieving better throughput.
3. The *Builder-Exchange* protocol (Section 4) is an extension of HotShot that allows its leaders to delegate the task of generating blocks to specialized entities called *builders*. This protocol achieves the "fair exchange" of the content of the block provided by the builder against the guarantee for this block to be committed to the chain.

As shown in Figure 1, the Espresso Sequencing Network interacts with users, builders, L2 rollups and the L1 blockchain.

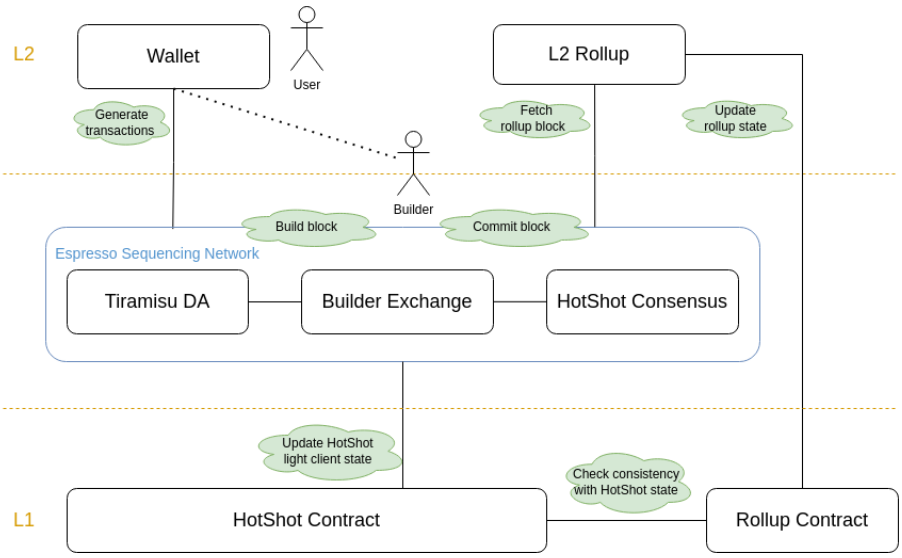

Figure 1: Main components in the HotShot protocol and highlevel workflow.

- **Users.** Users are participants who create transactions by signing some piece of data with their private key. These transactions can then be sent to the Tiramisu DA mempool which is public, or some other entity such as a private pool / builder.
- **Builders.** Builders have the ability to select and order transactions in order to create valuable blocks. They obtain users' transactions from the public mempool or some other potentially private source.
- **Layer 2 (L2) rollups.** These are off-chain execution engines (VMs) that accept users' transactions and deterministically process them after being ordered and finalized by the Espresso Sequencing Network. Their execution logic could be anything from app-specific to fully-featured smart contract platforms (like EVM rollups). Furthermore, the prover network as a subcomponent will periodically update the state commitments in the rollup contract on L1, along with a *validity proof* (for zkRollups) or a potential *fraud proof* (for optimistic rollups).
- **Layer 1 (L1) blockchain.** The Espresso Sequencing Network checkpoints the ledger state to this blockchain. Its primary function is to serve as an always-online, minimally trusted verification light client for the HotShot Consensus layer. When the L1 is more mature with wider adoption and higher economic cost for forking/reorg, these checkpoints also provide a defense in depth on long-range attacks on HotShot consensus. Internally, there is a *HotShot contract* that logs the finalized ledger produced by the HotShot sequencers; and one *rollup contract* per L2 rollup that reads the ledger state from the HotShot contract and maintains its rollup-specific states.

The high level workflow between all these components is made of "tasks". Some of these tasks are sequential while others are performed in parallel.

1. **Generate transactions.** Users through their wallets generate transactions which are then forwarded to some public mempool (e.g. Tiramisu DA mempool) or some private one, which can be managed or accessed by a builder. At this stage transactions have no specific order and do not belong to a block yet.
2. **Build block.** Builders continuously monitor transactions potentially coming from different sources with the goal of creating valuable blocks. Once the block is ready, the builder may participate in some kind of selection process (e.g. auction) involving a HotShot leader who has the right to append new blocks to the HotShot ledger.
3. **Commit block.** After this selection process is over, the builder and the HotShot leader engage in the *Builder Exchange* protocol which aim is to ensure that the block is committed while no information about its content is leaked to the leader. This interaction also involves the Tiramisu DA component that ultimately stores and disseminate the block data to all the replicas of the HotShot protocol.

4. **Fetch rollup block.** Rollups monitor the state of the HotShot ledger and periodically fetch the transactions corresponding to their namespace. By doing so they obtain a rollup block which can be derived deterministically from the contiguous sequence of Espresso Sequencing Network blocks and the rollup namespace.
5. **Update rollup state.** Armed with this block data and potentially some additional auxiliary information (e.g., snark proof), the rollup can update the state on its L1 contract in order to make all the transactions included in this block final.
6. **Update HotShot light client state.** Rollup state validation in the L1 rollup smart contract consists of a number of steps that includes ensuring the rollup state is consistent with the HotShot ledger state. While this can be done in a number of ways (e.g. verifying a zero-knowledge proof inside the rollup contract), the Espresso Sequencing Network contract gives access to already validated HotShot states to any L1 (rollup) contract.
7. **Check consistency with HotShot state.** As mentioned above, some rollups may decide to fetch the HotShot ledger state directly from the HotShot contract in order to verify that the rollup state is consistent with the consensus state.

3 Tiramisu Data Availability

3.1 Overview

In a conventional consensus protocol each node votes to finalize a new block only after it has seen the entire data payload for that block. If the payload has size $|B|$ then this requires $O(n\,|B|)$ communication which is a key barrier to throughput. A conventional approach to scaling would send the data only to a sub-committee. Unfortunately, this is not resilient against our assumed bribing adversary: if an adversary bribed the entire sub-committee holding copies to a finalized block, its payload is forever lost—a catastrophic failure of liveness.

The Espresso Sequencing Network forms agreement on an ordered list of *certificates of data availability (DA)* and therefore, it does not need to disseminate the entire block contents to nodes. In this section we describe *Tiramisu*—an efficient, three-layer solution to the DA problem. Each layer represents a point on the security-performance tradeoff curve. Importantly, the common case of Tiramisu is fast, keeping the full system (optimistically) responsive. (See Figure 2.)

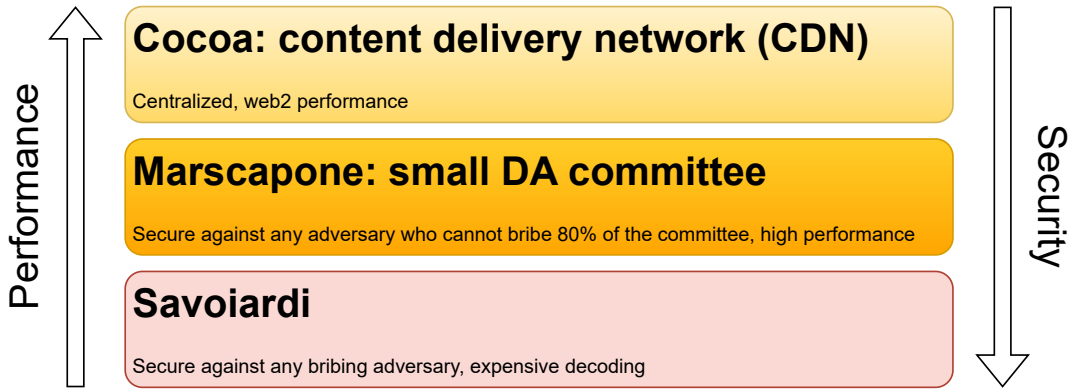

Figure 2: Tiramisu: a three-layer solution for data availability (DA).

Base layer. *Savoiardi: bribery-resilient DA.* In rare cases the network operates under *pessimistic* conditions: the network is under active attack by a powerful active or bribing adversary, and the other layers of Tiramisu might cease to function for the duration of this attack.

In this case we rely on a solution similar to Ethereum's Danksharding proposal [73]. The proposer of a new block encodes its payload under an erasure code. Then the proposer partitions the encoded payload into small shares and distributes only these small shares among all nodes in the network. Assuming $|B|$ is large enough, each piece has size $O(|B|/n)$ and the total communication is $O(|B|)$. Savoiardi introduces an enhancement to previous information dispersal schemes that adapts VID to HotShot's linear communication topology. A full description of Savoiardi is given in Appendix A.

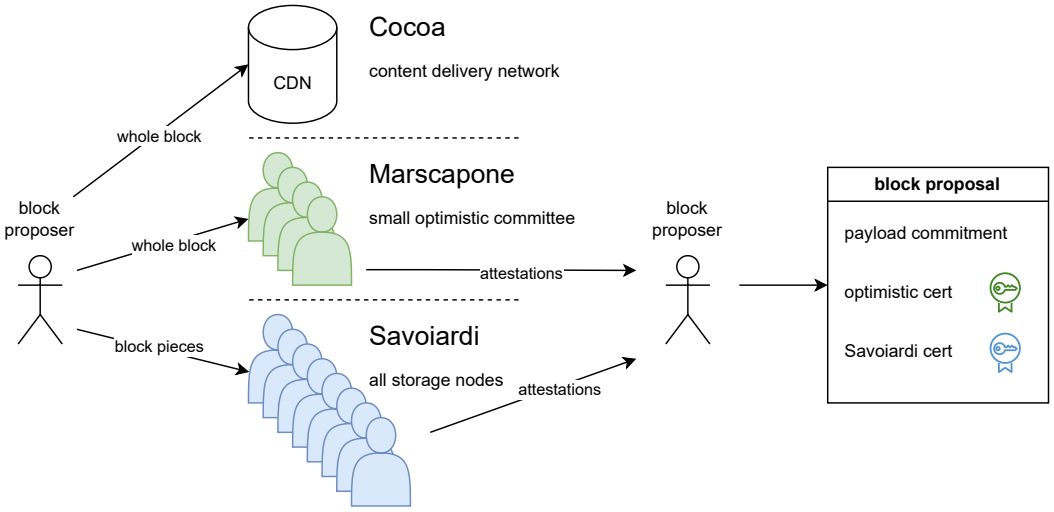

Figure 3: Tiramisu flow. Block proposer disperses block among validator nodes, aggregates attestations into certificates, builds a candidate HotShot block.

Security. This erasure-coded payload has the property that the entire payload can be recovered from any sufficiently large subset of shares. Thus, even if a powerful adversary corrupts many nodes, the remaining honest nodes are able to recover the payload.

Performance. A disadvantage of Savoiardi is that no single node has the entire payload. A client who wishes to recover the full payload must download shares from many nodes and spend computation resources to decode the payload. Thus, while Savoiardi is the ultimate defense against powerful adversaries, for performance reasons its use should be avoided in all but the worst network conditions.

Middle layer. *Mascarpone: a small DA committee for fast consensus.* Most of the time we expect the network to operate under *optimistic* conditions: the network is not under attack by a powerful active or bribing adversary.

In such cases the network can avoid Savoiardi's expensive payload recovery process and instead rely on a small, constant-size DA committee. The block proposer or other participant uploads a new block's payload to the committee, and the block is finalized after receiving attestations from a quorum of committee members[1].

Security. Members of the committee are selected at random, and the committee is replaced at the beginning of each epoch. Randomness for this selection is sourced from the decentralized random beacon (DRB) as described in Section B.1 of the HotShot whitepaper [84]. The size of the committee is chosen so that if a passive adversary corrupts less than 1/3 of all stake then at least one committee member is honest with overwhelming probability. Thus, the Mascarpone layer is secure against all but the most powerful adversaries. It can be compromised only by an adaptive adversary who can quickly corrupt the committee before it is replaced with a new one, or a bribing adversary who can corrupt the committee immediately.

Top layer. *Cocoa: Web2 performance from a CDN.* We can further improve performance under optimistic network conditions via a centralized content distribution network (CDN).

This solution is simple: the block proposer uploads a new block's payload to the CDN, and anyone who wants the block payload may request it from the CDN much like a Web2 streaming service consumer might download a video.

The CDN can also serve as an ultra-fast channel for passing messages between the block proposer and other nodes for use cases such as collecting attestations for the quorum certificate.

A centralized CDN might be an easy target for an attacker, or it could experience occasional downtime even in the absence of any malicious actor. As such, the Cocoa of a CDN is well-supported by the Mascarpone layer below it when the CDN us unavailable but network conditions are otherwise optimistic.

[1]This is the step that may fail in the presence of a bribing adversary, but will succeed in optimistic conditions as mentioned above.

3.1.1 Communication complexity

Tiramisu achieves asymptotically optimal $O(|B|)$ total network communication, summed over all three layers of the protocol:

Base layer (Savoiardi). The Savoiardi erasure-code dispersal scheme achieves $O(|B|)$ as described in Appendix A.6.

Middle layer (Mascarpone). The full block payload is sent to each member of the Mascarpone committee. The size of this committee is constant. (Say, 10–200 nodes.) As such, total network communication for this layer is $O(|B|)$.

Top layer (Cocoa). Like Mascarpone, the full block payload is sent to only a constant number of nodes. In Cocoa, the payload is sent to only a single node—the CDN. Thus, total network communication for this layer is $O(|B|)$.

3.2 How HotShot uses Tiramisu

HotShot's use of Tiramisu is simple. During each view, a HotShot consensus proposer (or its delegate, see Section 4) will attempt to finalize a commitment to a block B with an accompanying certificate of availability. A DA-certificate is obtained by the consensus proposer by utilizing Tiramisu. Given a block payload B, the proposer performs the following:

1. Compute the payload commitment $C \leftarrow \texttt{Commit}(B)$.
2. Initiate all three layers of Tiramisu concurrently:

   Savoiardi. Execute $\texttt{Savoiardi.Disperse}(B)$ with all nodes.

   Mascarpone. Upload (B, C) to the small DA committee.

   Cocoa. Upload (B, C) to the content delivery network (CDN).

Validator nodes vote in HotShot for a candidate block only if they have seen their own Savoiardi share of the encoded block. Under this scheme, if the HotShot vote passes then it must also be the case that a quorum of Savoiardi nodes has each seen its Savoiardi share, so there's no need to compile these attestations into a certificate.

In addition, nodes in the DA small sub-committee of Mascarpone respond with a signed certificate of C.

The proposer of the next block (or its delegate) awaits responses of both types above to compose a certificate of availability: votes attesting availability of Savoiardi pieces from of quorum of $2f + 1$ nodes, and attestations of C from a threshold of the small DA sub-committee. The block commitment C and its corresponding certification of DA, denoted $\texttt{cert}(C)$, are included on-chain. As discussed previously, the payload B is too big to fit on-chain, but the security properties of Tiramisu ensure that B is available.

3.2.1 Liveness

Recall that the block proposer waits for two types of responses to form a certificate of availability, a quorum of votes attesting availability of Savoiardi pieces and attestations of C from a threshold of the small DA sub-committee.

Savoiardi is guaranteed to succeed by the properties of the Savoiardi scheme and the HotShot threat model. But Mascarpone could fail if a sufficiently powerful adversary corrupts the optimistic DA committee so that it is unable to produce a certificate. In this case, no block can be finalized for this HotShot view—a temporary failure of liveness.

Because the optimistic DA committee is selected at random and frequently refreshed, the only way an adversary can reliably cause the committee to fail is to execute an expensive adaptive or bribery attack. As discussed in Section 5.1, these attacks exhaust the adversary's budget, after which liveness recovers immediately.

To put this attack in perspective, observe that the adversary could cause a similar liveness failure more cheaply by attacking only the *leader* for this view, so this avenue for attack cannot further weaken HotShot liveness.

3.2.2 Forcing expensive data recovery

Assume that the Cocoa layer of Tiramisu (CDN) is not functioning for some reason, as there can be no problem retrieving payload data when the CDN is functioning. Of all the layers of Tiramisu, Savoiardi is the only layer that is secure against an adaptive or bribing adversary. Thus, we expect there must exist an attack whereby the adversary forces the network to rely on the expensive Savoiardi scheme to recover the block payload.

The only such attack is to corrupt the small DA committee so that it produces a certificate, yet is unwilling or unable to deliver the payload upon request. In Section 3.2.1 we observed that liveness attacks exhaust the adversary's budget. The same principle applies to attacks that force expensive data recovery.

4 The Builder-Exchange

In this section, we describe an extension of HotShot that enables an ecosystem of block builders in the Espresso Sequencing Network by *delegating* the proposing task to external builders. Delegating proposing to external builder is consistent with the HotShot approach of keeping validators free of pre-validating transactions and therefore help promote extremely high performance. It allows builders to emerge and become highly specialized in specific rollups, languages, and MEV extraction. However, the separation of builder and proposer roles presents inherent challenges.

Builders invest effort into producing a block, e.g., identifying arbitrage opportunities and eploiting MEV, and they need protection from having their effort stolen. Conversely, each consensus proposer has a unique privilege to gain fees and block rewards that they do not want to miss, and need protecting from untrusted builders. In particular, a key challenge relates to a *fair exchange* between the builder and HotShot, where deciding the builder block is exchanged for a fee by the builder and for the builder's block content. To understand the concern better, consider several strawman scenarios.

If the builder sent the block to an intermediary, such as the consensus proposer or the DA sub-committee of the Espresso Sequencing Network, it must trust the intermediary. However, a bribing adversary might steal the builder effort and replace the block or simply censor it.

If the builder utilized a simple "commit-reveal" scheme (e.g., Fino [64], Ferveo [10], Suave [43]), letting consensus decide first on a builder's commitment, and then revealing the contents, then untrusted builders could spam the system with commitments and drop. This has two adverse affects. First, proposers may not be able to collect fees or rewards for their slots. Second, they would leave the chain with 'holes' in the sequence.

Conversely, if the builder paid fees upfront, then bad proposers might charge the block fee without including it in the sequence.

In fact, it should not be surprising that these scenarios present conflicting tradeoffs. The fair-exchange problem has been studied extensively in the cryptography literature, and it is known that even a mere fair-exchange of cryptographic strings among mistrusting parties cannot be solved deterministically [29]. Note that approaches based on the idea of gradually releasing a secret [12, 35, 21] do not work in our context due to tight timing constraints.

To strike a balance between the conflicting requirements, we devise a *Builder-Exchange* mechanism that protects both sides against all but the most implausible attack scenarios. In particular, our mechanism protects the builder from revealing its block against a bribing adversary who can control the proposer and an entire DA sub-committee, while assuming two-thirds of the validators are honest. It protects the consensus network from holes in the sequence, i.e., liveness it always maintained under our threat model. Additionally, HotShot will be able to retrieve the builder's block and make a decision assuming rational builders, who are eager to reveal their block once it is guaranteed a slot in the sequence. Note that an irrational builder could not send anything or send an invalid block in any case. Therefore, forcing a builder to reveal the block does not provide any benefit. Uniquely, the Espresso Sequencing Network supports paying fees that are predicated on a decision to include a block which in a pre-designated slot. This guarantees the system can charge the builder a fee even if the block itself is invalid.

Below we describe the Builder-Exchange solution in the context of HotShot as the underlying consensus protocol; however, the mechanism is suitable for other consensus protocols and may be of interest on its own.

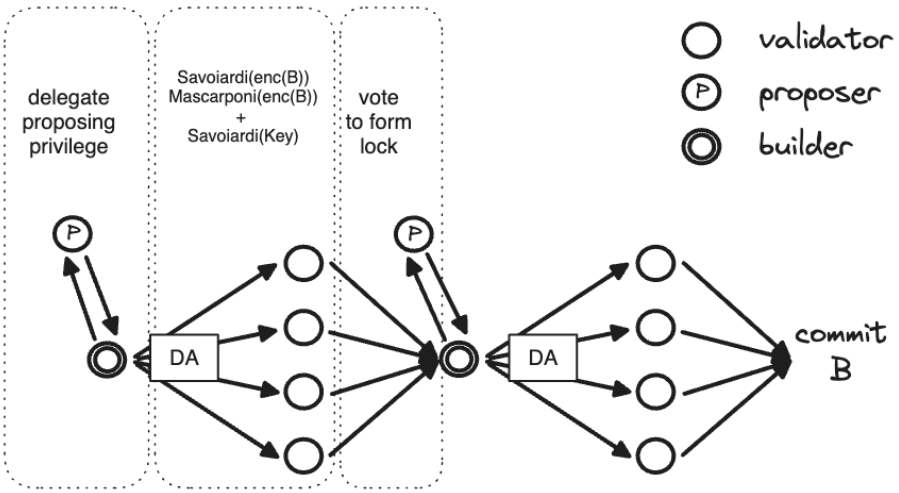

Figure 4: Builder-driven exchange between a builder and the consensus network.

The Builder-Exchange mechanism is a combination of two core ingredients: (i) a commit-reveal regime, and (ii) a Builder-driven extension to HotShot. Figure 4 depicts the flow of a block from the builder until a consensus decision to include it in the sequence:

The commit-reveal scheme is fairly standard: The builder sends an encrypted block proposal and uses secret-sharing to escrow the key with the validators. In this way, the builder is guaranteed that unless a third of the validators collude to reveal the key, the block contents cannot be revealed.

The builder also sends a fee payment predicated on a consensus decision that includes its block. In this way, if a majority of validators prevent the commit, they will simply be lose progress and be left without the fee.

The Builder-driven regime is somewhat unique. The builder itself obtains the lock and sends it to validators. Builder-driven lock dissemination guarantees that there will be $2f + 1$ validators that prevent the next leader from skipping this slot.

To maintain efficient retrieval of block data through Mascarpone or Cocoa, in this step the builder should also send the key K_v to the small Mascarpone sub-committee and publish it to the Cocoa CDN.

5 Full Protocol

In this section, we present an overview of the full HotShot protocol. The protocol revolves around a consensus core, modified to utilize Tiramisu for DA, as described in Section 3, and finally, incorporating a Builder-driven exchange between builders and the consensus core as explained in Section 4.

5.1 The Threat Model

For simplicity, most of this paper uses a static, permissioned setting consisting of n validator nodes ("nodes") out of which up to $f < n/3$ are Byzantine. However, as described in the Espresso Sequencing Network whitepaper [84], HotShot adapts a permissioned Byzantine Fault Tolerant (BFT) protocol to the proof-of-stake setting. This enables us to support dynamic network participants that can freely join and leave protocols by bonding or unbonding stake. The protocol should satisfy safety and liveness so long as more than two-thirds of the total amount of stake is controlled by honest nodes.

To achieve full decentralization, we need to support tens of thousands of staked consensus nodes. Traditional approaches usually combine a permissioned BFT protocol with proof-of-stake via committee sampling, where a small random committee will represent the entire set of staked users to run the consensus. However, this type of scheme usually suffers from adaptive attacks, where an adversary, that can control only a small amount of stake, can still corrupt the elected committee and break the security of consensus. There do exist solutions (e.g., Algorand [47], YOSO [46])

secure against adaptive adversaries. The best known defense is to hide the elected committee until they published their votes. Thus, an adaptive adversary cannot change the committee's behavior after the fact.

However, a bribery attack is still a practical concern for this solution, where a malicious adversary, even without knowing who to corrupt, can advertise payouts for certain verifiable malicious behaviors [9], e.g., the attacker can create a smart contract that pays elected committee members to censor specific transactions. An important aspect of modelling such adversaries is that they must have limited financial resources. Indeed, a bribing adversary with infinite budget is not realistic, and could simply cause a permanent liveness failure we cannot protect against. Thus, to achieve the desired efficiency and scalability without losing bribery resistance, we let all staked nodes participate in the consensus protocol. It is worth noting that this choice of adversarial model led to mandating a consensus core protocol whose steady state (optimistic) communication complexity is linear: HotStuff-2 [63] as the underlying BFT protocol, coupled with a view sync protocol ("pacemaker") from Naor and Keidar [70].

5.2 Views and proposers.

The HotShot protocol operates in a view-by-view manner. Each view v has a designated leader L_v, and an external party U_v elected as a builder. A discussion of the Espresso's lottery mechanism for managing builder selection is provided in [85] and is orthogonal to the discussion in this paper.

The protocol consists of two parts, a steady-state protocol and a view synchronization (or pacemaker) protocol for advancing views. To guarantee liveness, a pacemaker synchronizes nodes to overlap in each view for sufficiently long. In this paper, we only elaborate on driving commit decisions. We defer the details of the view synchronization to the HotShot whitepaper [84].

Block format. The protocol forms a chain of values. We use the term *block* to refer to each value in the chain. We refer to a block's position in the chain as its *height*. A block $\mathbf{B}_k$ at height k is chained to the block $\mathbf{B}_{k-1}$ preceding it using a cryptographic commitment denoted $\mathsf{Commit}(\mathbf{B}_{k-1})$. The form of commitment is determined by the DA layer. (The reason HotShot cannot use a simple hash of $\mathbf{B}_{k-1}$ as other blockchains is that $\mathbf{B}_{k-1}$ itself is not seen by all nodes, only the commitment used in the DA layer for information dispersal.)

The block at height k has the following format

$$\mathbf{B}_k := (b_k, h_{k-1})$$

where b_k denotes a proposed value at height k and $h_{k-1} := H(\mathsf{Commit}(\mathbf{B}_{k-1}))$ is a hash of the previous block's commitment. The first block $\mathbf{B}_0 = (b_0, \perp)$ has no predecessor. HotShot does not perform validity check on block content except for verifying that it is chained to a certificate of availability of its predecessor. We say $\mathbf{B}_l$ *extends* $\mathbf{B}_k$, if $\mathbf{B}_k$ is an ancestor of $\mathbf{B}_l$ ($l > k$).

Encrypted blocks by builders. When an elected builder U_v has a block b_v to propose, it initially hides the contents using a symmetric key K_v it encrypts $E_v \leftarrow \mathsf{Enc}_{K_v}(b_v)$. It then shares $b_k := (E_v, K_v)$ (we will see how below).

Certificates and certified blocks. In the protocol, nodes vote for blocks using an aggregate signature. To vote for a block $\mathbf{B}_k = (b_k, h_{k-1})$, a node signs $H(\mathsf{Commit}(b_k), h_{k-1})$. We use $C_v(\mathbf{B}_k)$ to denote a set of signatures by $2f + 1$ nodes in view v. We call $C_v(\mathbf{B}_k)$ a certificate or a quorum certificate (QC) for $\mathbf{B}_k$ from view v. Certified blocks are ranked by the views in which they are certified, i.e., a certificate $C_v(\mathbf{B}_k)$ is ranked higher than $C_{v'}(\mathbf{B}_{k'})$ if $v > v'$.

Locked blocks. At any time, each node locks the highest certified block to its knowledge. During the protocol execution, each node keeps track of all signatures for all blocks and keeps updating its locked block, and uses them to guard the safety of a commit.

View protocol. A view proceeds according to the following flow:

1. **Delegate (leader only).** At the beginning of each view, the leader waits to collect a QC from the immediately proceeding view, or for the pacemaker module (mentioned above) to *expire*

the previous view. The leader L_v sends the elected builder U_v a signed delegation certificate $D_v := \langle pubkey(U_v), h_{k-1} \rangle_{L_v}$, along with a certificate $C_{v'}(\mathbf{B}_{k-1})$ known to leader which h_{k-1} refers to. The builder should extend h_{k-1}. In steady state, h_{k-1} will refer to the immediately preceding view. If a QC from the immediately preceding view is not available, for instance, because the previous leader crashed, then the leader includes the QC from the highest view known to it.

2. **Propose (builder only).** The builder generates a block $\mathbf{B}_k \leftarrow (b_k, h_{k-1})$ extending the latest certified block it was given by the leader. It then uses the DA layer to spread two items: (i) it invokes Tiramisu (all three layers) to disseminate $(\mathsf{PROPOSE}, v, E_v, C_{v'}(\mathbf{B}_{k-1}))$ signed by U_v, along with the delegation certificate D_v and a commitment $C_{v,1} \leftarrow \mathsf{Commit}_{VID}(E_v)$. (ii) simultaneously, it invokes Savoiardi to disseminate K_v along with a commitment $C_{v_2} \leftarrow \mathsf{Commit}_{VSS}(K_v)$. The builder uses a variant of Savoiardi that shares K_v and its commitment $C_{v,2}$ with validators using a verifiable secret-sharing [27, 41] scheme. In practice, we use a variant of Feldman's scheme [41] where the linear size commitment to the polynomial is replaced by a KZG commitment [52]. More details can be found in Appendix B. Importantly, K_v is **not** broadcast via Mascarpone or Cocoa and each key share sent to replica R must be encrypted with R's public key. The builder also sends a fee payment predicated on a consensus decision that includes a block and key that match the commitment at the designated slot. Note that the Espresso Sequencing Network uniquely supports such predicated payments.

   Crucially, only the commitments and K_v shares incur a linear communication blowup, while the (encrypted) block content E_v is disseminated erasure-coded via the DA layer, and sent in full only to a small sub-committee. Hence, this protocol preserves the communication efficiency of Tiramisu.

3. **Vote (all nodes).** A node waits to receive the first proposal in view v signed by U_v carrying (i) a valid certificate D_v of delegation of L_v to U_v (ii) an erasure-coded piece of E_v matching a commitment $C_{v,1}$, (iii) a secret-share of K_v matching a commitment $C_{v,2}$, (iv) a QC $C_{v'}(\mathbf{B}_{k-1})$ ranked no lower than the locked block. To vote for the block $\mathbf{B}_k = (b_k, h_{k-1})$, the node signs $h_k := H(C_{v_1}, C_{v,2}, h_{k-1})$ and sends $\langle \mathsf{VOTE}, h_k, v \rangle$ as a threshold signature share to U_v. It updates lock to $C_{v'}(\mathbf{B}_{k-1})$.

4. **Drive (builder only).** Upon collecting $2f + 1$ vote shares, form $C_v(h_k)$ and broadcast to all nodes. To maintain efficient retrieval of block data through Mascarpone or Cocoa, in this step the builder should also send the key K_v to the small Mascarpone sub-committee and publish it to the Cocoa CDN.

Retrieving a block. Under (common) optimistic scenarios, retrieving the block content is done by obtaining E_v and K_v through the Cocoa or Mascarpone layers. In worst case scenarios, retrieving the block content is done by reconstructing the key K_v using Shamir secret-sharing, which is a linear transformation, and reconstructing the encrypted block E_v through Savoiardi. In either case, applying the key K_v to E_v decrypts the block. Importantly, even if the builder acts irrationally and sends a bogus key K_v, both E_v and K_v can be retrieved. That is, no hole will be left in the chain, the builder will be charged a fee for a unique (bogus) block. Finally, note that replicas must wait for the block $\mathbf{B}_k$ to be committed before sending the share of the corresponding symmetric key K_v in order to avoid a malicious participants to collect shares early on and steal the content of the block.

Committing a block. A block $\mathbf{B}_k$ is said to be committed if there exists an $l \geq k$ such that $C_{v'}(\mathbf{B}_l)$ and $C_{v'+1}(\mathbf{B}_{l+1})$ are formed, and $\mathbf{B}_l$ extends $\mathbf{B}_k$. In other words, either for $\mathbf{B}_k$ or for one of its successors, two blocks at consecutive heights are certified in consecutive views. The existence of these certificates guarantees that the encrypted content E_v of the block value b_v is retrievable from one of the Tiramisu layers as well as the corresponding decryption key K_v such that $E_v = \mathsf{Enc}_{K_v}(b_v)$.

Omitted Details. Several mechanisms are left outside the scope of this paper; the details are provided in [84]. Briefly, they include several efficient primitives (e.g., aggregated quorum certificates, stake table and decentralized random beacons) to adapt the permissioned settings to the fully decentralized setting without performance deterioration. The stake stable maps public keys of stake-holders to validators, and it is managed on the L1 mainnet. When the set of stakers change,

HotShot dynamically reconfigures the validator set. Additionally, as already mentioned, HotShot integrates a view synchronization protocol based on Naor-Keidar [70].

6 Related Work

The problem of state machine replication [59, 60, 79] studies how multiple deterministic distributed machines can agree on a common shared state, even if some of the machines are adversarial or *Byzantine*. The last four decades have seen a lot of progress towards designing Byzantine Fault Tolerant (BFT) protocols for SMR [23, 61, 68, 67, 47, 36]. While reviewing all the work in this space is beyond the scope of this manuscript, we will describe aspects that are closely related to HotShot.

Achieving high throughput and low latency. Several works in the past two decades have focused solely on designing protocols that can process a large number of transactions with low latency. This includes Nakamoto style protocols [69, 19, 39, 75, 76, 33, 56, 91, 94, 58] as well as classical BFT protocols under different network conditions such as synchrony [26, 3, 4, 38, 47, 86, 77, 80, 2], partial synchrony [23, 15, 93, 25, 24, 45, 88], and asynchrony [5, 48, 62, 82, 45]. To improve throughput, a recent line of work attempts to separate data dissemination from consensus on transactions [28, 37, 36, 82, 81, 76, 91]. Similarly, in terms of latency, the notion of optimistic responsiveness was introduced and shown to hold for many protocols [75, 77, 80, 93, 25, 23]; in a nutshell, a responsive protocol finalizes transactions at the speed of the network (independent of any pessimistic upper bound on network delay Δ).

HotShot borrows many of the above advances and its core consensus protocol is based on HotStuff-2 [93, 63].

Data availability. At a high level, data availability protocols must ensure that once the data is shared among the nodes of the network, it can be reliably recovered later. A naive way to achieve such a goal is to store this data directly on the ledger. In the context of Ethereum, this is the approach that was taken by most rollups before the introduction of Proto-Danksharding (EIP4488) [20], causing high gas cost and reduced throughput.

Similarly to Ethereum's Danksharding proposal [73], Tiramisu leverages erasure codes [90] and polynomial commitments [52] in order to share the data among replicas. This is how we achieve low communication complexity for disseminating a block B on the critical path: assuming $|B| \gg n$, where n is the number of parties, the total communication remains $O(|B|)$ as each party only receives a piece of data of size proportional to a chunk of the block. While recovery is expensive in the worst case, Tiramisu additionally offers the Mascarpone and Cocoa layers to address efficient retrieval in the optimistic scenarios.

Bribery resistance. Prior works have classified adversaries as either static, adaptive, or mobile. While static corruption is a weak adversarial assumption for open blockchain settings, practical solutions are not known under strong mobile adversaries [74, 95, 13, 42]. Thus, many works aim to secure themselves under an adaptive adversary, although many of them still incur a high communication complexity [14, 22, 57, 96, 3, 38, 1, 53]. Algorand [47] improves upon this to simultaneously achieve low communication complexity using small committees, in addition to security against an adaptive adversary. However, as also mentioned in [9], we note that an adversary can utilize bribery attacks to blindly corrupt parties, making solutions based on small committees insecure. Our solution, on the other hand, obtains the desirable performance parameters while being secure under an adaptive and bribing adversary.

Use of a CDN. We leverage a hybrid network composed by a Content Delivery Network (CDN) coupled with a P2P network. In the optimistic scenario, all messages and data will be exchanged through the CDN, thus guaranteeing close to optimal throughput and latency. While the idea of combining CDN and P2P networks to improve content distribution has been explored in previous works [97, 50], their purpose is to boost the performance of centralized systems. In our case the goal is to maximize network performance most of the time while retaining liveness and censorship resistance even when the CDN is disrupted or compromised.

MEV. Maximal Extractable Value (MEV) [34] is a serious concern as it may not only harm user experience but also undermines the security of the consensus protocol. Among the numerous approaches to mitigate the negative effects of MEV [64, 54, 49, 55, 87, 98, 44], Proposer-Builder-Separation (PBS) aims at allowing validators (a.k.a. proposers) to outsource the construction of blocks to a competitive builder market place. At the core of PBS is the fair exchange problem, where a builder sends the contents of its block and some payment to the proposer in the hope this block will be appended to the ledger. In practice the problem is solved via a trusted party called *Relay* [44] that allows builders and proposers to run the block auction and ensure both parties comply with their obligations. In addition to the strong trust assumption, currently relays are not economically sustainable [11] and thus can only be considered as temporary solution. Getting rid of the trust assumption would require to enshrine PBS [72] into the consensus protocol (e.g. Ethereum) which is a complex task and might make it difficult to implement a potentially better alternative in the future.

Our Builder-Exchange protocol leverages the idea of delegation to circumvent the complexity of standard approaches such as *two-slots PBS* [18]. Another benefit from our solution is its modularity, as it can be integrated to any auction or builder selection protocol. More generally, our Builder-Exchange mechanism enables the evolution of an ecosystem that manages MEV.

References

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected $o(1)$ rounds, expected $o(n^2)$ communication, and optimal resilience. Cryptology ePrint Archive, Paper 2018/1028, 2018. https://eprint.iacr.org/2018/1028.

[2] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected o (1) rounds, expected communication, and optimal resilience. In *International Conference on Financial Cryptography and Data Security*, pages 320–334. Springer, 2019.

[3] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, and Ling Ren. Dfinity consensus, explored. *Cryptology ePrint Archive*, 2018.

[4] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 106–118. IEEE, 2020.

[5] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2019.

[6] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Asynchronous verifiable information dispersal with near-optimal communication, 2022. https://eprint.iacr.org/2022/775.

[7] Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. Succinct erasure coding proof systems, 2021. https://eprint.iacr.org/2021/1500.

[8] Guillermo Angeris, Alex Evans, and Tarun Chitra. A note on bundle profit maximization, 2021.

[9] Vivek Bagaria, Amir Dembo, Sreeram Kannan, Sewoong Oh, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Proof-of-stake longest chain protocols: Security vs predictability. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, pages 29–42, 2022.

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

[11] The Block. Blocknative suspending mev-boost relay to focus on 'economically viable opportunities', 2023. https://www.theblock.co/post/253035/blocknative-suspending-mev-boost-relay-to-focus-on-economically-viable-opportunities , Accessed 2024-04-13.

[12] Dan Boneh and Moni Naor. Timed commitments. In *Annual international cryptology conference*, pages 236–254. Springer, 2000.

[13] Silvia Bonomi, Antonella Del Pozzo, Maria Potop-Butucaru, and Sébastien Tixeuil. Optimal mobile byzantine fault tolerant distributed storage. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 269–278, 2016.

[14] Gabriel Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162, 1984.

[15] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.

[16] Vitalik Buterin. Reed-solomon erasure code recovery in $n \log^2 n$ time with ffts, August 2018. https://ethresear.ch/t/reed-solomon-erasure-code-recovery-in-n-log-2-n-time-with-ffts.

[17] Vitalik Buterin. A rollup-centric ethereum roadmap, 2020. https://ethereum-magicians.org/t/a-rollup-centric-ethereum-roadmap/4698, Accessed: 2024-04-14.

[18] Vitalik Buterin. Two-slot proposer/builder separation, 2021. https://ethresear.ch/t/two-slot-proposer-builder-separation/10980, Accessed: 2024-04-14.

[19] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.

[20] Vitalik Buterin, Dankrad Feist, Diederik Loerakker, George Kadianakis, Matt Garnett, Mofi Taiwo, and Ansgar Dietrichs. Eip-4844: Shard blob transactions," ethereum improvement proposals, no. 4844, 2022. `https://eips.ethereum.org/EIPS/eip-4844`, Accessed: 2024-04-11.

[21] Philippe Camacho. Fair exchange of short signatures without trusted third party. In *Topics in Cryptology–CT-RSA 2013: The Cryptographers' Track at the RSA Conference 2013, San Francisco, CA, USA, February 25-March 1, 2013. Proceedings*, pages 34–49. Springer, 2013.

[22] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 42–51, 1993.

[23] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 173–186. USENIX Association, 1999.

[24] Benjamin Y Chan and Rafael Pass. Simplex consensus: A simple and fast consensus protocol. *Cryptology ePrint Archive*, 2023.

[25] TH Hubert Chan, Rafael Pass, and Elaine Shi. Pala: A simple partially synchronous blockchain. *Cryptology ePrint Archive*, 2018.

[26] TH Hubert Chan, Rafael Pass, and Elaine Shi. Pili: An extremely simple synchronous blockchain. *Cryptology ePrint Archive*, 2018.

[27] Benny Choc, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Annual Symposium on Foundations of Computer Science (Proceedings)*, pages 383–395, 1985.

[28] Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Matteo Monti, and Manuel Vidigueira. Every bit counts in consensus. *arXiv preprint arXiv:2306.00431*, 2023.

[29] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 364–369, 1986.

[30] Coindesk. Ethereum bot gets attacked for $20m as validator strikes back. `https://www.coindesk.com/business/2023/04/03/ethereum-mev-bot-gets-attacked-for-20m-as-validator-strikes-back/`, Accessed: 2024-04-07.

[31] Cointelegraph. Sandwich trading bots lose bread and butter in $25m exploit, 2023. `https://cointelegraph.com/news/sandwich-trading-bots-lose-bread-and-butter-in-25m-exploit`, Accessed: 2024-04-14.

[32] Cointelegraph. zksync went down for 5 hours on christmas day but is now back online, 2023. `https://cointelegraph.com/news/zksync-went-down-5-hours-christmas-day-now-back-online`, Accessed: 2024-04-14.

[33] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*, pages 23–41. Springer, 2019.

[34] 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 *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

[35] Ivan Bjerre Damgård. Practical and provably secure release of a secret and exchange of signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 200–217. Springer, 1993.

[36] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.

[37] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. Cryptology ePrint Archive, Paper 2021/777, 2021. `https://eprint.iacr.org/2021/777`.

[38] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[39] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. {Bitcoin-NG}: A scalable blockchain protocol. In *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, pages 45–59, 2016.

[40] Dankrad Feist and Dmitry Khovratovich. Fast amortized kzg proofs, 2023. `https://eprint.iacr.org/2023/033`.

[41] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.

[42] Orr Fischer and Merav Parter. Distributed congest algorithms against mobile adversaries. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, pages 262–273, 2023.

[43] Flashbots. The future of mev is suave. `https://writings.flashbots.net/the-future-of-mev-is-suave/`, Accessed: 2023-05-11.

[44] Flashbots. Mev boost, 2024. `https://docs.flashbots.net/flashbots-mev-boost/introduction`, Accessed 2024-04-13.

[45] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pages 296–315. Springer, 2022.

[46] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. Yoso: You only speak once: Secure mpc with stateless ephemeral roles. In *Annual International Cryptology Conference*, pages 64–93, 2021.

[47] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

[48] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous bft protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 803–818, 2020.

[49] Lioba Heimbach and Roger Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, AFT '22. ACM, September 2022.

[50] Hai Jiang, Jun Li, Zhongcheng Li, and Xiangyu Bai. Efficient large-scale content distribution with combination of cdn and p2p networks. *International Journal of Hybrid Information Technology*, 2(2):4, 2009.

[51] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium*, pages 1353–1370, 2018.

[52] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.

[53] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In *Annual International Cryptology Conference*, pages 445–462. Springer, 2006.

[54] Alireza Kavousi, Duc V. Le, Philipp Jovanovic, and George Danezis. Blindperm: Efficient mev mitigation with an encrypted mempool and permutation. Cryptology ePrint Archive, Paper 2023/1061, 2023. https://eprint.iacr.org/2023/1061.

[55] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40*, pages 451–480. Springer, 2020.

[56] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference*, pages 357–388. Springer, 2017.

[57] Valerie King and Jared Saia. Breaking the o (n 2) bit barrier: scalable byzantine agreement with an adaptive adversary. *Journal of the ACM (JACM)*, 58(4):1–24, 2011.

[58] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th usenix security symposium (usenix security 16)*, pages 279–296, 2016.

[59] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21:558–565, 1978.

[60] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16:133–169, 1998.

[61] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, 2001.

[62] Yuan Lu, Zhenliang Lu, and Qiang Tang. Bolt-dumbo transformer: Asynchronous consensus as fast as the pipelined bft. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2159–2173, 2022.

[63] Dahlia Malkhi and Kartik Nayak. Hotstuff-2: Optimal two-phase responsive bft. *Cryptology ePrint Archive*, 2023.

[64] Dahlia Malkhi and Pawel Szalachowski. Maximal extractable value (mev) protection on a dag. *arXiv preprint arXiv:2208.00940*, 2022.

[65] Ori Mazor and Ori Rottenstreich. An empirical study of cross-chain arbitrage in decentralized exchanges. Cryptology ePrint Archive, Paper 2023/1898, 2023. https://eprint.iacr.org/2023/1898.

[66] Conor McMenamin. Sok: Cross-domain mev, 2023.

[67] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. Cryptology ePrint Archive, Paper 2016/199, 2016. https://eprint.iacr.org/2016/199.

[68] Iulian Moraru, David G Andersen, and Michael Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 358–372, 2013.

[69] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260, 2008.

[70] Oded Naor and Idit Keidar. Expected linear round synchronization: The missing link for linear byzantine smr, 2020. https://arxiv.org/abs/2002.07539.

[71] Kamilla Nazirkhanova, Joachim Neu, and David Tse. Information dispersal with provable retrievability for rollups. *arXiv preprint arXiv:2111.12323*, 2021.

[72] Mike Neuder. Why enshrine proposer-builder separation? a viable path to epbs, 2023. https://ethresear.ch/t/why-enshrine-proposer-builder-separation-a-viable-path-to-epbs/15710 , Accessed 2024-04-13.

[73] Valeria NikolaenkoDan and Dan Boneh. Data availability sampling and danksharding: An overview and a proposal for improvements. https://a16zcrypto.com/posts/article/

an-overview-of-danksharding-and-a-proposal-for-improvement-of-das/, Accessed 2023-07-12.

[74] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks. In *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 51–59, 1991.

[75] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. Cryptology ePrint Archive, Paper 2016/917, 2016. https://eprint.iacr.org/2016/917.

[76] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *Proceedings of the ACM symposium on principles of distributed computing*, pages 315–324, 2017.

[77] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37*, pages 3–33. Springer, 2018.

[78] Polygon. Aggregated blockchains: A new thesis. https://polygon.technology/blog/aggregated-blockchains-a-new-thesis, Accessed: 2024-04-11.

[79] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[80] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the optimality of optimistic responsiveness. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 839–857, 2020.

[81] Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. Shoal: Improving dag-bft latency and robustness, 2023.

[82] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.

[83] Starkware. Starknet. https://www.starknet.io/, Accessed: 2023-05-11.

[84] Espresso Systems. The espresso sequencer: Hotshot consensus and tiramisu data availability, 2023. https://github.com/EspressoSystems/HotShot/blob/main/docs/espresso-sequencer-paper.pdf, Accessed: 2024-04-7.

[85] Espresso Systems. The espresso market design, 2024. https://hackmd.io/n0z0sBkNS3irfttkeG5xzw.

[86] DFINITY Team et al. The internet computer for geeks. *Cryptology ePrint Archive*, 2022.

[87] Shutter team. Combating front-running and malicious mev using threshold cryptography. https://blog.shutter.network/, Accessed: 2024-04-13.

[88] TD Team et al. State machine replication in the diem blockchain, 2021.

[89] Barry Whitehat. zkrollup, 2018. https://github.com/barryWhiteHat/roll_up, Accessed: 2023-05-11.

[90] Stephen B Wicker and Vijay K Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[91] Lei Yang, Vivek Bagaria, Gerui Wang, Mohammad Alizadeh, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Scaling bitcoin by 10,000 x. *arXiv preprint arXiv:1909.11261*, 2019.

[92] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. DispersedLedger: High-Throughput byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 493–512, Renton, WA, April 2022. USENIX Association.

[93] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

[94] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. Ohie: Blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 90–105. IEEE, 2020.

[95] Moti Yung. The" mobile adversary" paradigm in distributed computation and systems. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 171–172, 2015.

[96] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 931–948, 2018.

[97] Ge Zhang, Wei Liu, Xiaojun Hei, and Wenqing Cheng. Unreeling xunlei kankan: Understanding hybrid cdn-p2p video-on-demand streaming. *IEEE Transactions on Multimedia*, 17(2):229–242, 2014.

[98] Liyi Zhou, Kaihua Qin, and Arthur Gervais. A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. *arXiv preprint arXiv:2106.07371*, 2021.

A Savoiardi Verifiable Information Dispersal

HotShot uses a variant of a VID scheme due to Alhaddad-Duan-Varia-Zhang (ADVZ) that those authors call *AVID-1* [7]. Our variant of AVID-1 is called *Savoiardi* and differs from the original scheme in the following ways:

1. Reduce the asymptotic communication burden of AVID-1 `Disperse` from quadratic to linear in the number of storage nodes. The quadratic communication of AVID-1 `Disperse` is due to the all-to-all messaging among storage nodes in the "echo" and "ready" steps of that scheme. HotShot eliminates these steps, thus achieving linear communication for `Disperse`. (See Section A.7 for discussion.)
2. Augment `Commit`(B) to include a constant-size vector commitment to certain polynomial evaluations as described below. The purpose of this augmentation is to enable the use of quasi-linear algorithms to batch-compute KZG proofs. (See Section A.5 for discussion.)

Next, we informally describe Savoiardi—see Algorithm 5 for pseudocode. Let n be the number of storage nodes, let r be the rate of the erasure code (example: $r = 1/4$), let $m = rn$ be the number of fragments into which the block payload B is split. Without loss of generality we assume that the block payload B consists of a list of scalars in some suitable prime field, and the size of this list is a multiple of m so that B has size km for some k.

A.1 Commit

View the block payload B as k sublists of m scalars each. For $i = 1, \ldots, k$ view the m scalars of sublist i as coefficients for a degree-$(m-1)$ polynomial p_i and let $\hat{p}_i$ denote the KZG [52] commitment to p_i. For each $j = 1, \ldots, n$ let

$$e_j = (p_1(j), \ldots, p_k(j))$$

denote the k-tuple of evaluations of these polynomials at j. Let vc denote an arbitrary constant-size vector commitment scheme. `Commit`(B) is defined as the pair (h, v) where

$$h = \text{hash}(\hat{p}_1, \ldots, \hat{p}_k)$$
$$v = \text{vc}(e_1, \ldots, e_n)$$

(If desired, the bit length of `Commit`(B) could be further reduced by hashing the pair (h, v).)

A.2 Disperse

`Disperse`(B) is a one-round interactive protocol between the block sender and the storage nodes. For $j = 1, \ldots, n$ the sender sends the following data to storage node j:

1. Polynomial commitments $\hat{p}_1, \ldots, \hat{p}_k$ and the vector commitment v. (This data is the same for each storage node.)
2. An evaluation tuple $e_j = (p_1(j), \ldots, p_k(j))$ and a vc opening v_j for e_j to v.
3. A constant-size aggregate KZG witness w_j of the polynomial evaluations relative to the polynomial commitments.

`Disperse`(B)

▷ Sender

1 : $p_1, \ldots, p_k \leftarrow$ interpret B as polynomials
2 : $\hat{p}_1, \ldots, \hat{p}_k \leftarrow$ KZG commitments to $p_1, \ldots, p_k$
3 : **for** $j = 1, \ldots, n$
4 : $\quad e_j \leftarrow (p_1(j), \ldots, p_k(j))$ evaluate polynomials
5 : **endfor**
6 : $h \leftarrow \text{hash}(\hat{p}_1, \ldots, \hat{p}_k)$
7 : $v \leftarrow \text{vc}(e_1, \ldots, e_n)$
8 : $t \leftarrow$ hash-to-field(h, v)
9 : $p \leftarrow \sum_{i=1}^{k} t^i p_i$ (random lin combo)
10 : $(w_1, \ldots, w_n) \leftarrow$ batch-KZG-prove(p)
11 : **for** $j = 1, \ldots, n$
12 : $\quad v_j \leftarrow$ open v at e_j
13 : $\quad$ Send to storage node j:
14 : $\quad\quad \hat{p}_1, \ldots, \hat{p}_k, v$ and e_j, w_j, v_j
15 : **endfor**

▷ Storage node j

16 : Receive $\hat{p}_1, \ldots, \hat{p}_k, v$ and e_j, w_j, v_j
17 : Verify vector opening v_j of v at e_j
18 : $h \leftarrow \text{hash}(\hat{p}_1, \ldots, \hat{p}_k)$
19 : `Commit`$(B) \leftarrow (h, v)$
20 : $t \leftarrow$ hash-to-field(`Commit`(B))
21 : $\hat{p} \leftarrow \sum_{i=1}^{k} t^i \hat{p}_i$
22 : $p(j) \leftarrow \sum_{i=1}^{k} t^i p_i(j)$
23 : KZG-verify$(\hat{p}, j, p(j), w_j)$
24 : Store $\hat{p}_1, \ldots, \hat{p}_k, v$ and e_j, w_j, v_j
25 : $\quad$ indexed by `Commit`(B)
26 : Send to Sender: sign(`Commit`(B))

▷ Sender

27 : Wait for q valid sigs $s_1, \ldots, s_q$
28 : $\quad$ of message `Commit`(B) from storage nodes
29 : $s \leftarrow$ aggregate sigs $s_1, \ldots, s_q$
30 : **return** certificate of retrievability
31 : $\quad cert($`Commit`$($`B`$)) = (s,$ `Commit`$(B))$

`Commit`(B)

1 : $p_1, \ldots, p_k \leftarrow$ interpret B as polynomials
2 : $\hat{p}_1, \ldots, \hat{p}_k \leftarrow$ KZG commitments to $p_1, \ldots, p_k$
3 : $h \leftarrow \text{hash}(\hat{p}_1, \ldots, \hat{p}_k)$
4 : **for** $j = 1, \ldots, n$
5 : $\quad e_j \leftarrow (p_1(j), \ldots, p_k(j))$ evaluate polynomials
6 : **endfor**
7 : $v \leftarrow \text{vc}(e_1, \ldots, e_n)$
8 : **return** commitment (h, v)

`Retrieve`$(c, cert(c))$

▷ Client

1 : Check validity of $cert(c)$
2 : Retrieve $\hat{p}_1, \ldots, \hat{p}_k, v$ from somebody
3 : $h \leftarrow \text{hash}(\hat{p}_1, \ldots, \hat{p}_k)$
4 : Verify $c = (h, v)$
5 : $t \leftarrow$ hash-to-field(h, v)
6 : $\hat{p} \leftarrow \sum_{i=1}^{k} t^i \hat{p}_i$
7 : Send to all storage nodes: "retrieve $c, cert(c)$"

▷ Storage node j

8 : Receive "retrieve $c, cert(c)$"
9 : Retrieve e_j, w_j, v_j and send to Client

▷ Client

10 : Receive e_j, w_j, v_j from storage node j
11 : Verify vector opening v_j of v at e_j
12 : $p(j) \leftarrow \sum_{i=1}^{k} t^i p_i(j)$
13 : KZG-verify$(\hat{p}, j, p(j), w_j)$
14 : Store j, e_j
15 : Retrieve from m storage nodes $j_1, \ldots, j_m$
16 : **for** $i = 1, \ldots, k$
17 : $\quad p_i \leftarrow$ Interpolate from $p_i(j_1), \ldots, p_i(j_m)$
18 : **endfor**
19 : $B \leftarrow$ Interpret $p_1, \ldots, p_k$ as a block payload
20 : **return** B

Figure 5: VID

The KZG witnesses $w_1, \ldots, w_n$ are computed as follows:

1. Compute the pseudorandom scalar

$$t = \text{hash-to-field}(\texttt{Commit}(B)) \tag{1}$$

2. Compute the polynomial p as a pseudorandom linear combination

$$p = \sum_{i=1}^{k} t^i p_i. \tag{2}$$

3. Each w_j is a KZG witness for the polynomial evaluation $p(j)$. Batch-compute all KZG witnesses $w_1, \ldots, w_n$ in quasi-linear time via the Feist-Khovratovich algorithm [40].

On receiving this data from the sender, each storage node j checks the integrity of its data. If the integrity check succeeds then the storage node stores its data for later use and replies to the sender with a signature of $\texttt{Commit}(B)$ to indicate its success.

The integrity check proceeds as follows for storage node j:

1. Verify the vc opening v_j for e_j relative to v.
2. Compute t as in (1) and the commitment $\hat{p}$ and evaluation $p(j)$ according to

$$\hat{p} = \sum_{i=1}^{k} t^i \hat{p}_i \tag{3}$$

$$p(j) = \sum_{i=1}^{k} t^i p_i(j) \tag{4}$$

3. Run KZG verification to check that the witness w_j is consistent with $\hat{p}$, j, and $p(j)$.

The sender waits for signatures of $\texttt{Commit}(B)$ from at least q storage nodes. (The choice of q is discussed in Section A.4.) The certificate of retrievability for block payload B consists of an aggregation of these q signatures and $\texttt{Commit}(B)$.

A.3 Retrieve

$\texttt{Retrieve}(c, \mathit{cert}(c))$ is a one-round interactive protocol between the client and storage nodes. The client fetches the polynomial commitments $\hat{p}_1, \ldots, \hat{p}_k$ and vector commitment v from somewhere—possibly from one of the storage nodes—and checks correctness of these commitments by verifying $c = (\text{hash}(\hat{p}_1, \ldots, \hat{p}_k), v)$. Next, the client computes the scalar t as per (1) and polynomial commitment $\hat{p}$ as per (3).

The client extracts the identities of at least q storage nodes from $\mathit{cert}(c)$ and sends a request to each such storage node for its block data for commitment c. Storage node j retrieves its data tuple e_j and witnesses w_j, v_j and sends this data to the client.

On receiving this data from a storage node j, the client checks the integrity of the data:

1. Verify the vector opening v_j for e_j with respect to v.
2. Compute $p(j)$ as per (4) and verify the KZG-witness w_j with respect to $\hat{p}$, j, and $p(j)$.

The client waits for at least m successful retrievals from storage nodes $j_1, \ldots, j_m$. For each $i = 1, \ldots, k$ the client recovers the degree-$(m-1)$ polynomial p_i from the m evaluations $p_i(j_1), \ldots, p_i(j_m)$ via interpolation. The coefficients of $p_1, \ldots, p_k$ are precisely the data in the block payload B.

A.4 Storage quorum size

The number q of storage nodes in a certificate of retrievability is chosen so that the `Disperse` sender and `Retrieve` client are both guaranteed to succeed even in the presence of up to f malicious storage nodes. Thus, we require $m + f \leq q \leq n - f$. There are many choices of f, m, q that meet this constraint. For example, the overhead from erasure encoding is inversely proportional to the erasure code rate $r = m/n$, which is maximized at $m = n - 2f$, implying $q = n - f$. Alternatively, a smaller choice of r enables larger f or smaller q.

A.5 On the need for a vector commitment

We defined `Commit`(B) in Section A.1 to include a commitment v to the vector $(e_1, \ldots, e_n)$ of polynomial evaluation tuples. Why? For each j the pseudorandom scalar t must depend on the evaluations $p_1(j), \ldots, p_k(j)$ as otherwise a malicious sender could produce a valid KZG witness for incorrect evaluations.

An alternative that avoids the need for a vector commitment is to define a different scalar t_j for each storage node j as $t_j = \text{hash}(\texttt{Commit}(B), p_1(j), \ldots, p_k(j))$ and compute p, $\hat{p}$, and $p(j)$ differently for each storage node using t_j instead of t.

Unfortunately, computation of these polynomials (and their KZG proofs) precludes the use of Feist-Khovratovich and introduces a quadratic dependence on the number n of storage nodes for the sender's run time. As quasi-linear runtime is a priority for HotShot, we prefer the additional communication overhead of the vector opening v_j over $O(n^2)$ run time for the sender.

A.6 Asymptotic complexity

Let $|B|$, $|open|$ denote the size of the block payload B and vector openings v_j, respectively. Total communication over all nodes for the payload B (without overhead) is $O(|B|)$. Overhead per node is $O(k + |open|)$. Recall that kn is $O(|B|)$. Thus, if $|open|$ is constant then Savoiardi achieves optimal asymptotic communication complexity $O(|B|)$.

Asymptotic computational complexity for both `Disperse` and `Retrieve` includes many costs, such as computation and verification of a vector commitment. But these costs are dominated by the discrete Fourier transforms (DFTs) computed in these protocols. Field arithmetic is cheaper than group arithmetic, so we account for these two costs separately.

Disperse. For each $i = 1, \ldots, k$ the polynomial evaluations $p_i(1), \ldots, p_i(n)$ cost $O(n \log n)$ for a total cost of $O(|B| \log n)$ field operations. The batch KZG proof costs $O(n \log n)$ group operations.

Retrieve. The k polynomial interpolations $p_1, \ldots, p_k$ each cost $O(n \log^2 n)$ [16] for a total cost of $O(|B| \log^2 n)$ field operations.

	Field ops	Group ops
`Disperse`	$\|B\| \log n$	$n \log n$
`Retrieve`	$\|B\| \log^2 n$	1

Table 1: Dominant DFT costs of two main procedures in Savoiardi.

A.7 Minimal termination guarantee

Why does AVID-1 `Disperse` of Ref. [7] have "echo" and "ready" steps, and why can these steps be safely eliminated in HotShot? These steps are necessary to achieve strong termination guarantees for AVID-1. Specifically, AVID-1 achieves both *termination* (if the sender is honest then all honest storage nodes complete `Disperse`) and *agreement* (if any honest storage node completes `Disperse` then all honest storage nodes complete `Disperse`).

As observed in Ref. [71], such strong termination guarantees are not needed in protocols such as HotShot. Instead, it suffices that only an honest sender for `Disperse` is guaranteed to complete the protocol and obtain a valid retrievability certificate . This weaker guarantee can be achieved without all-to-all messaging among storage nodes and so we may safely eliminate these steps in HotShot.

A.8 Strong availability guarantee

Some VID protocols offer only a weak availability guarantee: if an honest client initiates `Retrieve`$(C, cert(C))$ then eventually it terminates and obtains some block payload B'. However, there is no guarantee that `Commit`$(B') = C$. This weak availability guarantee allows for a much simpler and faster VID protocol that can be instantiated with any erasure code and any hash function. Examples of state-of-the-art VID protocols of this type include AVID-M [92] and the unnamed protocol of Ref. [6].

The weak availability guarantee implies that a maliciously dispersed payload might not be discovered during `Disperse`. Instead, discovery must wait until `Retrieve`, where the client can re-compute $\texttt{Commit}(B')$ to check consistency with commitment C.

In the event where a HotShot adversary corrupts *both* the VID `Disperse` sender *and* the entire HotShot optimistic DA committee so that the DA committee stores no data and the retrieved block payload B' is inconsistent with the commitment C, the data-availability can be lost. Fortunately, the event that $\texttt{Commit}(B') \neq C$ can be an evidence for identifying and penalizing a corrupt `Disperse` sender.

- Example: someone could assemble a subset S of storage node shares that recovers B' and create SNARK proof that S is inconsistent with C.
- Example: A quorum of storage nodes could each attest that B' is consistent with its own share but inconsistent with C.

However, the above mitigations are complex and expensive. A complex, expensive, and rarely-used mitigation process is especially vulnerable to mistakes. It is not clear that the performance benefits of weak availability VID is worth this risk.

A.9 Related work

As mentioned previously, Savoiardi is a variant of AVID-1 [7] with weaker termination guarantees. A similar state-of-the-art protocol is *Semi-AVID-PR* due to Nazirkhanova-Neu-Tse [71]. Note that we can alternatively use *Semi-AVID-PR* as our VID protocol, which has features like a transparent setup and the support to fast discrete-log-based curves. The tradeoff is that *Semi-AVID-PR* has higher verification/communication complexity compared to AVID-1 when the number of storage nodes is large.

In Section A.8 we cited Refs. [92, 6] as examples of state-of-the-art VID protocols with a weak availability guarantee.

B Verifiable Secret Sharing

Verifiable Secret Sharing [27] (VSS) is a cryptographic primitive that allows a dealer P_d (in our context, the builder) to share a secret value s (symmetric key K_v) among n players $P_1, P_2, \cdots, P_n$ (replicas) so that $t < n$ shares do not leak any information about the secret s, while $t+1$ shares are enough for its recovery. We follow the scheme described in [52] (see Section 4.1) that essentially replaces the linear size commitment in Feldman's construction [41] with a (constant size) polynomial commitment.

Share Phase. In this phase, shares are computed, distributed and verified.

- P_d computes a t-degree polynomial $Q(X) := s + \sum_{i=1}^{t} r_i X^i$ where r_i are randomly sampled.
- P_d computes the polynomial commitment to $Q(X)$, $\tilde{Q} := \mathsf{KZG.Commit}(pp, Q(X))$, where $pp := \mathsf{KZG.Setup}()$ is some public parameter. In our setting, $\tilde{Q} = \mathsf{Commit}_{VSS}(s)$.
- P_d sends $\langle \mathsf{SHARE}, \tilde{Q}, i, Q(i), w_i \rangle$ to each replica P_i where $w_i := \mathsf{KZG.CreateWitness}(pp, Q(X), i)$ is the polynomial evaluation witness.
- Each replica then runs $\mathsf{KZG.VerifyEval}(pp, \tilde{Q}, i, Q(i), w_i)$ and accepts if the verification succeeds.

Recovery Phase. In this phase the shares are collected, verified and combined to recover the secret.

- A participant R willing to recover the secret s broadcasts a message $\langle \mathsf{RECOVER}, v \rangle$ where v is a view number.
- When receiving this message, all honest replicas check first that the corresponding secret s for view v corresponds to a committed block, otherwise an error message is returned.
- Each replica P_i then sends $\langle \mathsf{SHARE}, \tilde{Q}, i, Q(i), w_i \rangle$ back to R.

- R waits for obtaining $t+1$ valid shares, that is shares that satisfies $\mathsf{KZG.VerifyEval}(pp, \tilde{Q}, i, Q(i), w_i) = \top$. If f is the number of corrupt nodes, we set the threshold t such that $f \leq t < n - f$, in order to ensure recovery is always possible.
- Using polynomial interpolation, R recovers $s = Q(0)$.