Data Availability Sampling with Repair

DAN BONEH, Stanford University, USA
JOACHIM NEU, a16z Crypto Research, USA
VALERIA NIKOLAENKO, a16z Crypto Research, USA
ADITI PARTAP, Stanford University, USA

Data availability sampling (DAS) is an important technique to horizontally scale consensus protocols without compromising on the number of adversarial nodes that can be tolerated. DAS is on the technical roadmap of major blockchains such as Ethereum. A major challenge for DAS schemes, that has not been formally studied in the literature, is how incomplete shares can be *repaired*. The need for repairing data shares motivates key aspects of Ethereum's DAS-based sharding vision called "Danksharding". In this work, we make two contributions. First, we provide a new definitional framework that formalizes the notion of repair, along with the security guarantees that a DAS scheme must provide. Second, we propose a new DAS scheme designed with efficient repair in mind, based on locally-correctable multiplicity codes. To facilitate using these codes, we introduce a new multivariate polynomial commitment scheme that (i) supports efficient openings of partial derivatives of a committed polynomial, (ii) supports fast batch opening proof generation at many points, and (iii) has an algorithm to recompute (repair) opening proofs at a point from only a few other proofs. The proposed scheme improves upon the state-of-the-art Ethereum Fulu DAS scheme, slated for deployment in late 2025/early 2026, in storage overhead, repair bandwidth and coordination, while only slightly increasing dispersal cost and sampling bandwidth. Our techniques readily carry over to data availability schemes based on verifiable information dispersal (VID).

1 Introduction

A key problem in blockchains is the design of *horizontally scalable* consensus protocols. These are consensus protocols where the transaction throughput increases (linearly) as new nodes are added to the system, while remaining secure as long as at most a constant fraction of the nodes are adversarial. A widely used technique to support horizontal scaling is to decouple the *dissemination* of transaction payloads from their *ordering*, by moving full block dissemination off the critical path of consensus. In many classical consensus protocols (Fig. 1(a)) nodes wait to receive a proposed block *in full* from the leader, before they vote for the proposal. In contrast, in this new paradigm (Fig. 1(b)), ordering consensus is established only over cryptographic *references* to payloads, and nodes only vote for a proposed reference to a payload once they are confident that the referenced payload is deemed *available*, meaning that in the future, anyone will be able to retrieve the payload associated with the reference. This approach has been studied in the literature [1, 12, 18, 46, 49, 52, 66] and can now be found in systems such as Ethereum [52], and NEAR [3].

This leads to an important primitive that one may call *verifiable distributed storage* (*VDS*) (Fig. 1(b), Fig. 2). We first describe VDS in isolation, and then circle back to discuss how it interacts with a consensus protocol to realize the aforementioned paradigm. VDS is provided collectively by a set of *storage nodes*. VDS has three key functionalities: *Dispersal* allows a client called *dispersor* to deposit a payload with the storage nodes for storage. *Retrieval* allows a client called *retrievor* to obtain a payload from the storage nodes, using a reference to the payload. *Verification* allows a client called *verifior* to check that indeed the payload associated with a given reference is available for retrieval from the storage nodes. In tandem with a consensus protocol for ordering references to payloads, the nodes in the consensus protocol also act as storage nodes in VDS. Each leader node of the consensus protocol acts as the dispersor and disperses

Authors' Contact Information: Dan Boneh, Stanford University, Stanford, CA, USA, dabo@cs.stanford.edu; Joachim Neu, a16z Crypto Research, New York, NY, USA, jneu@a16z.com; Valeria Nikolaenko, a16z Crypto Research, New York, NY, USA, valeria.nikolaenko@gmail.com; Aditi Partap, Stanford University, Stanford, CA, USA, aditi712@cs.stanford.edu.

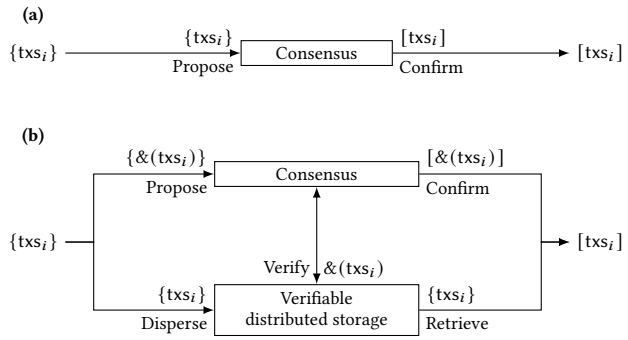


Fig. 1. (a) Traditionally, consensus protocols order full transaction payloads (cf: from an unordered set $\{\mathsf{txs}_i\}$ to a sequence $[\mathsf{txs}_i]$). (b) In the new paradigm for horizontally scalable consensus, dissemination and ordering of payloads are decoupled. Consensus operates only on references to payloads (cf: $\&(\mathsf{txs}_i)$), while the full payload is made available through a verifiable distributed storage system (VDS). Before voting for a reference proposed in consensus, nodes verify that the underlying payload is available for retrieval.

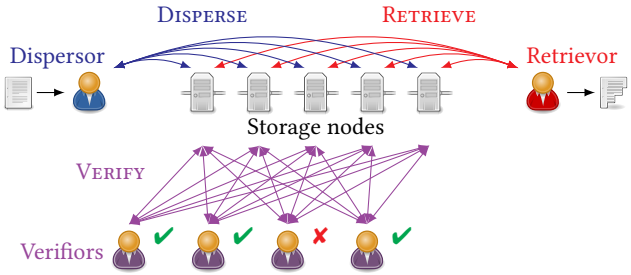


Fig. 2. Verifiable distributed storage (VDS): Dispersal allows a dispersor to deposit a payload with the storage nodes for storage. Retrieval allows a retrievor to obtain a payload from the storage nodes, using a reference to the payload. Verification allows verifiors to check that indeed the payload associated with a given reference is available for retrieval from the storage nodes. Vote-based data availability (VDA) and data availability sampling (DAS) differ in the methods used during verification.

to the VDS the full payload associated with a proposed reference. During consensus, when nodes validate a proposal for consensus, they verify with the VDS that the payload is indeed available. This ensures that whatever references to payloads are confirmed in consensus, every client can later act as a retrievor, and obtain (from the VDS) the payloads associated with the ordered references. This way the client recovers the overall sequence of payloads that are the output of the consensus protocol.

The perspective of VDS unifies and highlights the commonalities between what one may call *vote-based data availability* (VDA) and *data availability sampling* (DAS), two dominant currents in contemporary scalable consensus design. Let us recall what these are, how they fit into the interface of VDS, and how they compare. Both VDA and DAS use erasure-correcting codes during dispersal and retrieval to provide fault tolerance while reducing overhead in terms of communication and storage. They differ in how they enable *verification*.

VDA originates in the academic literature on verifiable information dispersal (VID) [10, 12, 33, 49, 66] and is used in systems like EigenDA [15], rollup data availability committees [14, 16], and Aptos QuorumStore [39]. During dispersal, every storage node acknowledges (with a signature) the receipt of its coded chunk of the payload — this acknowledgment is effectively a *vote* for the chunk. A quorum of such acknowledgments acts as a *non-interactive* availability proof for

the payload. Verification of a reference's availability boils down to verifying the signatures in the proof. Security rests on the assumption that if many signatures attest to the availability of a payload, and the adversary cannot control too many of the storage nodes, then many honest storage nodes must be left that have indeed stored their coded chunks of the payload, and from which the payload is available for retrieval. Advantages of VDA include simplicity, and a straightforward point-to-point network model. A drawback is that VDA requires an honest majority assumption among storage nodes [49].

In DAS on the other hand, verification is *interactive*. Every verifior samples the storage nodes to "see for itself" how many got their coded chunks. It is often assumed that the verifior's sampling queries are unlinkable (meaning that the adversary does not know which queries come from which verifior) and the adversary's responses to sampling queries are non-adaptive [36] (meaning, before learning verifiors' queries, the adversary has to choose which chunks to release; this is the so-called "blackboard sampling model" [50] or "strong model"). Under these assumptions, the verifior gets a reliable estimate as to whether enough coded chunks are available to allow retrieval of the payload in the future. An advantage of DAS is the promise of security even under adversarial majority. Drawbacks of DAS include that it imposes stringent unlinkability/anonymity requirements on the network layer, which are not satisfied by current protocols [50], and doubts remain whether *any* protocol can efficiently satisfy them [36, 50].

Nonetheless, DAS is at the core of Ethereum's scaling roadmap, with visions such as "Danksharding" [52], and concretely specified protocol extensions like Fulu DAS [23] which is slated for deployment in late 2025/early 2026. Improving DAS is the goal of this paper. Despite our focus on DAS, our techniques transfer readily to VDA.

Because the blackboard sampling model is notoriously hard to instantiate, recent works have focused on an interpretation of DAS as "collective security." The idea is that if enough (honest) verifiors reach the conclusion that a payload is available, then the payload can actually be retrieved *from the samples collected by the verifiors*. Importantly, this property can be achieved in the "weak network model" of point-to-point communication where queries are linkable to their originating verifiors, and with an adversary that is adaptive to sampling queries. This reconstruction property was formalized as an extractor-based definition in a recent work [31].

Our Contributions. We build on the work of [31], improve it in a number of ways, and propose a new DAS scheme.

First, as a "housekeeping" item, we extend the security definition of [31], to capture the large pool of verifiors. In more detail, [31] analyzes the probability of being able to decode if a *given* set of m verifiors accepts a payload as available. But, it does not account for the fact that the overall pool of sampling verifiors may be much larger than m, and the adversary only needs to answer the samples of a subset of these verifiors, to make them believe that the payload is available. This gives the adversary more power since it may adaptively select which subset of m verifiors' samples it answers. The effect of this needs to be reflected in the security definition. The required extra union bound over m-size subsets leads to increased error probability, and thus should be made explicit.

Second, we introduce new security properties for a successful payload retrieval, while both [31] and [52] say little in that regard. We make retrieval explicit in the problem formulation and state the desired security properties. Specifically, we define two notions: the first captures a worst-case retrieval from verifiors' samples (this is based on the extractor-based definition of [31], and works even under a dishonest majority of storage nodes). The second notion is a new security property for optimistic-case retrieval from storage nodes in the common case when many storage nodes are honest.

Third, and more importantly, we formalize the notion of repair for storage nodes. We observe that a key motivation for the proposal of 2D Reed–Solomon erasure codes for Ethereum *Danksharding*'s vision [52] is that it would be easier to *repair* the coded chunks of storage nodes that have for some reason not received their chunks during regular dispersal.

In contrast, a simple 1D Reed–Solomon based scheme, such as the *Fulu DAS* design [23] slated for implementation in Ethereum in late 2025/early 2026, has to fully decode and re-encode the payload to repair missing chunks. As was observed in [63], this is a costly process that currently requires beefy altruistic "supernodes" that replicate the entire payload. We capture the syntax and desiderata of repair in our problem formulation. Being able to do repair efficiently is crucial for the security of the system, for otherwise an adversarial dispersor could grieve the system and impact its performance by not dispersing properly (but still "good enough" to let a payload be accepted).

Finally, we propose a new construction that achieves efficient local repair while greatly reducing the size of chunks stored by the storage nodes. We identify that the repair problem maps to erasure-correcting codes that are locally-correctable (LCCs [67]). Hence, we base our construction on LCCs called *multiplicity codes* [38], which are a generalization of Reed–Solomon and Reed–Muller codes, in the sense that besides evaluations of a polynomial they also include evaluations of derivatives of it. We demonstrate that this construction provides 4.3× improved node storage, 2× reduced number of subnets contacted along with 2.1× lower bandwidth usage for repair, as compared to Ethereum's Fulu DAS proposal. The cost is a 1.5× increase in multi-threaded dispersor runtime. Note that while the sampling bandwidth is 1.5× higher, Ethereum's proposal does not achieve the same level of security as our scheme. See Tabs. 1 and 2 for more details.

To be able to employ these multiplicity codes, *we develop a new polynomial commitment scheme.* This is the first commitment scheme for multivariate polynomials that supports i) opening derivatives of the polynomial at any point, ii) computing batch openings efficiently, and iii) repairing opening proofs at a point from only a few other proofs (as [19, 70] do for KZG commitments and [71] for multivariate commitments). For instance, to open a bivariate polynomial and both its first derivatives at any point, the proof size for our scheme is only three group elements, and the verifier work is dominated by four pairings. This can be useful for other applications as well, such as verifiable secret sharing [70].

We fully implement our encoding and commitment scheme in C to show that they are indeed efficient in practice. Specifically, the time to encode and commit to a part of a payload is 46.1ms and 44ms respectively. Moreover, verifying an opening proof takes about 2.9ms. While the batch opening algorithm takes about 20s in sequential compute runtime, we note that it is highly parallelizable, and can be executed in just 0.38s on a machine with 96 cores. We stress that according to Ethereum's latest DAS designs [26], calculating a blob's encoding and opening proofs can take place off the critical path of consensus. Furthermore, Ethereum blocks are produced every 12s, and many rollups post blobs at an even lower frequency. As a result of these observations, we do not believe that the computational load is prohibitive.

1.1 Technical Overview

We now provide an overview of our contributions.

1.1.1 Formal definitions. We extend the definitions of [31] to formally model the notions of data dispersal, retrieval, and repair. Fig. 3 gives an overview of our syntax—the dispersor computes a commitment which acts as the reference, along with an encoding of the payload, and then disperses it among the storage nodes. A verifior can run the Verify protocol to query the storage nodes for random positions of the encoding, to see if the payload is indeed available. Any storage node can run the Repair protocol to repair its chunk of the encoding. Lastly, a retrievor can run the Retrieve protocol to retrieve the encoded payload.

We define two security notions. The first is called *worst-case soundness*. It generalizes the extractor-based definition from [31] to our model, which says that if enough verifiors believe that the payload is available, then it is possible to extract the payload from all the samples collected by the verifiors. The second is a new security property, which we call *optimistic-case soundness*. It captures the idea that if enough verifiors agree that the data is available, and if a large

number of storage nodes are honest, then it should be possible to retrieve the full payload by running the Retrieve protocol. In other words, a DAS scheme that satisfies optimistic-case soundness, guarantees that the stored payload is retrievable *from the storage nodes*, if the adversary is not too powerful. To compare different schemes in terms of their security, we look at sampling bandwidth, which refers to the total bandwidth usage of a verifior so as to achieve a certain level of (i) worst-case soundness, and (ii) optimistic-case soundness, assuming an upper bound on how many storage nodes the adversary is allowed to corrupt. We formally define these notions in Sec. 2.

Additionally, to make the notions of dispersal and repair meaningful, we impose efficiency requirements on the protocols. Specifically, we require that the amount of data stored by each storage node, i.e. the chunk size, must be much smaller than the size of the full payload. This is to avoid the trivial solution where each node has to store the full encoding. Similarly, there is always a trivial Repair protocol which simply runs Retrieve to retrieve the payload, and re-encodes it to compute a chunk. However, such an approach would beat the whole point of defining a Repair protocol. Hence, we impose efficiency requirements on Repair, ensuring that coordination over the network for repair remains low and that the total bandwidth usage is minimized. We formalize these requirements in Sec. 2.

A key concern with DAS has been, how, from a peer-to-peer networking point of view, sampling can be implemented efficiently and effectively in practice [50]. Ethereum's Fulu DAS [23, 26] uses GossipSub [45, 61], a flooding-based peer-to-peer publish-subscribe protocol. GossipSub maintains topics called "subnets" to which peers can *subscribe* in order to receive all messages that other peers *publish* to that topic. An honest node that receives a message on a topic, forwards said message to its peers that are subscribed to the same topic. This "broadcast channel with relaying" gossip network can thus be modelled as satisfying the property that if some honest node receives a message, then shortly thereafter, all honest nodes interested in that message will have received it. Equivalently, the only way to withhold a message from *some* honest node is to withhold it from *all* honest nodes. We make this gossip network assumption in DAS explicit, and analyze the security of both Fulu DAS and our constructions in this model.

1.1.2 New constructions. We present new DAS constructions that achieve efficient local repair. For our construction with batching, a node only needs to join 32 other subnets to repair its chunk, in contrast to 64 in the Ethereum Fulu DAS proposal (and 867 in [31] and 8192 in [30]). Moreover, for a single blob of size 128 KB, the total bandwidth used for repair is only 60 KB, as compared to 128 KB for Ethereum. We achieve this while requiring nodes to store 4.3× less data as compared to Ethereum (but more than in [30, 31]). Note that our sampling bandwidth is slightly larger, but Ethereum's design does not achieve the same level of optimistic-case soundness as our scheme. While the dispersor work is larger, we note that (i) the dispersors in Ethereum today would be the block builders, which are very powerful already because they participate in MEV extraction (see [65] and references therein), and hence this may not be a problem at all, (ii) moreover, the dispersor work for our scheme is highly parallelizable—we discuss this more in Sec. 4.1 and Sec. 5. Tabs. 1 and 2 give a detailed comparison between our scheme, the Ethereum proposal and other existing schemes including FRIDA [30] and [31].

At a high level, we rely on a new generic framework for constructing DAS schemes from erasure codes and commitment schemes. This generalizes the framework presented in [31] to our security definitions. Informally, it uses an erasure code to encode the payload, and commits using an erasure code commitment corresponding to the same code (formally defined in [31]). Specifically, an erasure code commitment allows committing to a message such that the encoding of this message can be opened at any position. Moreover, any set of openings produced by a computationally bounded adversary is guaranteed to be consistent with at least one codeword from the erasure code. In the DAS framework, each storage node stores one position of the encoding, along with a corresponding opening proof for

Table 1. Comparing our proposed DAS construction with the Ethereum Fulu DAS proposal for distributing a single blob of size 128 KB. For local repair, we compare two metrics: i) the number of gossip subnets that a particular node needs to contact to repair its chunk, ii) the total bandwidth usage of running the Repair protocol. Note that, we compare repair during the dispersal/sampling phase wherein a gossip protocol is used at the network layer (see Secs. 2 and 3 for details). For comparing the security guarantees across the schemes, we compare the total bandwidth used by a sampling client, to achieve a 40 bits of optimistic-case soundness, assuming that upto 20% of the nodes can be corrupt, and 40 bits of worst-case soundness (both formally defined in Sec. 2). For these calculations, we assume that there are 10^4 verifiors, and sampling is considered successful if any 5 percent of the verifiors get valid responses (these parameters are based on prior analysis by Ethereum Foundation researchers [62]). Note that we compare to schemes in [30, 31] separately in Tab. 2 because we did not find any numbers for these schemes for data of size 128 KB.

Metric	**Ethereum Fulu DAS [23]**	**Our construction** (*vs. Fulu DAS*)	**Our construction** with batching (Sec. 4.3)
Node storage	8.18 KB (256 F + 4 $\mathbb{G}$)	0.23 KB (3 F + $\mathbb{G}$) (35× *better*)	1.88 KB (24 F + $\mathbb{G}$) (4.3× *better*)
Local repair: total bandwidth	128 KB	21.3 KB (273 F + $\mathbb{G}$) (6× *better*)	60 KB (2.1× *better*)
Local repair: number of subnets	64	91 (1.4× *worse*)	32 (2× *better*)
Sampling bandwidth	16.37 KB (512 F + 8 $\mathbb{G}$)	9.14 KB (117 F + $\mathbb{G}$) (1.8× *better*)	24.4 KB (312 F + $\mathbb{G}$) (1.5× *worse*)
Optimistic-case soundness (in bits)	0	40	40
Multi-threaded dispersor runtime†	0.3s	0.42s (1.4× *worse*)	0.47s (1.5× *worse*)
Number of subnets for dispersal	128	9216	1225
Commitment size	48 B (1 $\mathbb{G}$)	48 B (1 $\mathbb{G}$)	384 B (8 $\mathbb{G}$)
Transparent setup	No	No	No

† When run serially, our implementation of the dispersor (without batching) takes 20s. But, the algorithm is highly parallelizable (see Fig. 12; whereas Fulu DAS's is essentially sequential). We estimate that the version without and with batching can run in 0.42s and 0.47s, respectively, on a machine with 96 cores (see Sec. 5.2 for details).

the commitment. Then, a verifior can probe the storage nodes for random positions of the codeword, and check the responses with respect to the commitment. Lastly, the payload can be extracted from the probes by running the decoding algorithm of the code. We describe this compiler and prove its soundness in Sec. 3.

The key insight that enables efficient local repair in our constructions is the use of locally correctable codes (LCC) to encode the data. LCCs are error-correcting codes wherein individual positions of the codeword can be recovered by querying only a few other positions of the codeword. This means that a storage node can recover the position in its chunk by interacting with only a few other storage nodes (or only a few other subnets, in the gossip network model).

We present constructions based on multiplicity codes, which are a generalization of Reed-Solomon and Reed-Muller codes. Informally, the payload data is interpreted as a multivariate polynomial. The encoding includes the evaluations

of this polynomial, and additionally, the evaluations of the partial derivatives of this polynomial. The derivatives essentially provide additional redundancy to the code, hence allowing for better distance, meaning we can decode the data from fewer positions, as well as efficient local correction. We discuss the design in more detail in Sec. 4.1. To disperse this encoding, each storage node stores one position of the codeword and in the gossip network model, a different subnet (or topic) can be used to disperse each codeword position. We assume that the number of storage nodes is a multiple of the length of the codeword. Our two constructions differ slightly in how the data is encoded—for the first construction, we interpret the data as a single polynomial. For the second construction with batching, we split the data into sub-blocks and encode each sub-block separately using multiplicity code. This leads to different tradeoffs—see Tabs. 1 and 2 for details. We discuss the batching design in detail in Sec. 4.3.

Additionally, we construct an erasure code commitment for multiplicity codes. This is the first multivariate polynomial commitment scheme that supports i) computing batch openings, i.e. we can efficiently compute the evaluation proofs for a set of points, ii) proving evaluations of derivatives, and iii) repairing the evaluation proof for a position given only a few other evaluation proofs. At a high level, we generalize the PST multivariate commitment [51] to allow checking the evaluation of the polynomial as well as its derivatives at a given point. Recall that the PST commitment relies on the fact that if a bivariate polynomial $f(X_0, X_1)$ evaluates to y at a point (a_0, a_1), then, there must exist quotient polynomials q_0, q_1 such that:

$$f(X_0, X_1) = (X_0 - a_0)q_0(X_0, X_1) + (X_1 - a_1)q_1(X_1) + y$$

To also verify the evaluation of the derivatives, we observe that at the point (a_0, a_1), the evaluations of the polynomials q_0, q_1 as defined above, are equal to the first-order derivatives. More formally, we have:

$$q_0(a_0, a_1) = f'_{x_0}(a_0, a_1) \quad \text{and} \quad q_1(a_1) = f'_{x_1}(a_0, a_1)$$

Here, f'_{x_0} and f'_{x_1} denote the first derivative of f with respect to x_0 and x_1, respectively. These observations were shown for univariate polynomials in [11], and we extend their proofs to multivariate polynomials in Sec. 4.2. Hence, to prove the evaluation of f and its first order derivatives at (a_0, a_1), the prover needs to compute the quotient polynomials q_0, q_1, along with their evaluation proofs at the same point. We also present efficient algorithms for computing batch openings (called one-to-many proofs in [70, 71]). These algorithms efficiently compute the evaluation proofs for a set of points. We do so by generalizing the techniques of [19] to multivariate polynomials. Moreover, we show how we can repair the evaluation proof of one point, given a few other evaluation proofs. We describe the full commitment scheme and its analysis in more detail in Sec. 4.2. We also implement our encoding and commitment scheme in C and show that it is indeed efficient. Tab. 1 compares our designs to the Ethereum Fulu DAS proposal with respect to efficiency and security guarantees. While our batch opening algorithm is more expensive, the structure of our encoding and commitment allows us to efficiently parallelize this work. We discuss this more in Sec. 4.1 and Sec. 5.

Additional related work is discussed in Sec. 7.

2 Definitions

In this section, we present the formal definition of a secure data availability scheme.

Notation. We use $\mathsf{A}(x)\langle \mathsf{S}(\mathsf{s}), \mathsf{R}(\mathsf{r})\rangle$ to denote an interactive protocol called A between parties S and R, wherein x denotes a global input known to all parties, and s and r are the inputs to S and R, respectively. We denote vectors by boldface lower-case letters, e.g. $\boldsymbol{u}$.

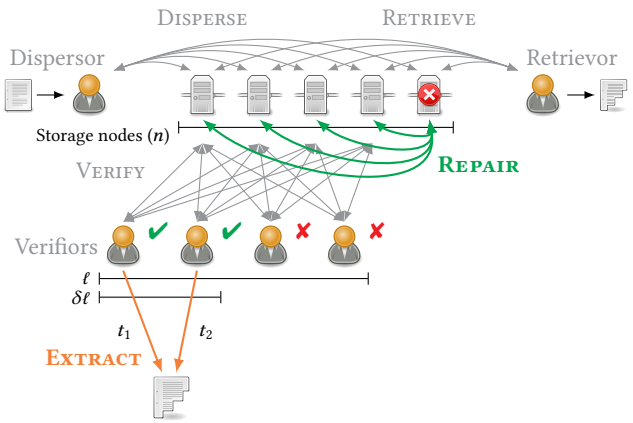


Fig. 3. Architecture of a data availability sampling (DAS) scheme as a concrete instantiation (and extension) of a VDS scheme (cf. Fig. 2). The dispersor encodes the payload and distributes it among the n storage nodes. The ℓ verifiors query the nodes to verify if the payload is indeed available. An *extraction* algorithm Ext allows to recover the payload from transcripts of any $\delta\ell$ verifiors that have deemed the data available. Later on, a retrievor can interact with the nodes to try to retrieve the payload. Lastly, any storage node can *repair* its chunk of the encoding using the Repair protocol.

2.1 Syntax

A Data Availability Sampling scheme DAS is a protocol between a dispersor D, n storage nodes denoted by $\mathsf{N}_1, \ldots, \mathsf{N}_n$, a verifior V and a retrieval client R. It consists of the following six PPT protocols:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ is a probabilistic algorithm that outputs the public parameters for the system. All the following algorithms and protocols implicitly take pp as input.
- $\mathsf{Disperse}(\mathsf{data}, n) \to (\mathsf{com}, d_1, \ldots, d_n)$ is a probabilistic dispersal algorithm run by the dispersor which outputs a commitment com to data, and data chunks d_i to be stored by the ith storage node, for all $i \in [n]$. We also define a sub-algorithm $\mathsf{Commit}(\mathsf{data}) \to \mathsf{com}$ that takes data as input, and outputs a commitment. Note that Commit is a deterministic algorithm since we do not need hiding commitments for data availability. We denote by $\mathsf{N}_i(d_i, \cdot)$ an honest storage node that is initialized with d_i data chunk, and can be queried during the Verify, Retrieve or Repair protocols.
- $\mathsf{Verify}(\mathsf{com})\langle \mathsf{V}, \mathsf{N}_1, \ldots, \mathsf{N}_n\rangle \to (b, \mathsf{tran})$ is a probabilistic sampling protocol between a verifior V and the storage nodes, to determine a bit that denotes whether the data is indeed available, along with a transcript containing all the queries made to the storage nodes and their responses.
- $\mathsf{Ext}(\mathsf{com}, \mathsf{tran}_1, \ldots, \mathsf{tran}_\ell) \to$ data OR $\perp$ is a deterministic extraction algorithm, which takes as input a commitment to the data along with a list of sampling transcripts $\mathsf{tran}_1, \ldots, \mathsf{tran}_\ell$, and outputs a valid opening data or $\perp$.
- $\mathsf{Retrieve}(\mathsf{com})\langle \mathsf{R}, \mathsf{N}_1, \ldots, \mathsf{N}_n\rangle \to$ data OR $\perp$ is a probabilistic reconstruction protocol, wherein a retrievor interacts with the storage nodes in an attempt to reconstruct the data.
- $\mathsf{Repair}(\mathsf{com}, i)\langle \mathsf{N}_1, \ldots, \mathsf{N}_n\rangle \to d_i$ is a probabilistic local reconstruction protocol. Any storage node $i \in [n]$ can execute this protocol with other nodes, to reconstruct its share of the data d_i in case it is lost or erased. Note that there are two scenarios where repair can be useful: (i) during dispersal or sampling: if the (potentially malicious) dispersor does not disperse the chunk of party i, or, (ii) after dispersal: the data has been dispersed but node i loses its chunk. We focus on the first scenario for the rest of the paper.

Table 2. Comparing our proposed DAS construction with the Ethereum Fulu DAS proposal and other schemes [30, 31], for distributing data of size 3 MB – we chose this based on the fact that Ethereum is planning to include 24 blobs with each block, with a total size of 3 MB by December 2025 [57]. All the metrics are as defined in Tab. 1.

Metric	**Ethereum Fulu DAS [23]**	**Hash [31]**	**FRIDA [30]**	**Our construction** (*vs. Fulu DAS*)	**Our construction** with batching (Sec. 4.3)
Node storage	196.5 KB	3.47 KB	2.72 KB	5.6 KB (24×3 F + $\mathbb{G}$) (*35× better*)	45 KB (24×24 F + $\mathbb{G}$) (*4.3× better*)
Local repair: total bandwidth	3 MB	3 MB	3 MB	511.9 KB (24×273 F + $\mathbb{G}$) (*6× better*)	1440 KB (*2.1× better*)
Local repair: number of subnets	64	867	8192	91 (*1.4× worse*)	32 (*2× better*)
Sampling bandwidth	393 KB	38.2 KB	168.6 KB	219.4 KB (24×117 F + $\mathbb{G}$) (*1.8× better*)	585 KB (312 F + $\mathbb{G}$) (*1.5× worse*)
Optimistic-case soundness (in bits)	0	40	40	40	40
Multi-threaded dispersor runtime†	0.3s	Implementation not available	Implementation not available	0.42s (*1.4× worse*)	0.47s (*1.5× worse*)
Number of subnets for dispersal	128	3468	32768	9216	1225
Commitment size	1.12 KB (24 $\mathbb{G}$)	443.9 KB	321.6 KB	1.12 KB (24 $\mathbb{G}$)	9 KB (24×8 $\mathbb{G}$)
Transparent setup	No	Yes	Yes	No	No

† Note that we report the runtime for one blob. When run serially, our implementation of the dispersor (without batching) takes 20s. But, the algorithm is highly parallelizable (see Fig. 12; whereas Fulu DAS's is essentially sequential). We estimate that the version without and with batching can run in 0.42s and 0.47s respectively, when run on a machine with 96 cores (see Sec. 5.2).

We now turn to the properties that DAS should satisfy.

Completeness. We say that a DAS scheme is complete if, whenever the dispersor honestly encodes and disperses the data, then the sampling protocol Verify returns 1, even if a certain fraction of the storage nodes are corrupt. This is formally defined in Def. 2.1.

Definition 2.1. We say that a Data Availability Sampling scheme $\mathsf{DAS} = (\mathsf{Setup}, \mathsf{Disperse}, \mathsf{Verify}, \mathsf{Retrieve}, \mathsf{Ext}, \mathsf{Repair})$ is f-correct if, for all λ, the following probability is negligible in λ:

$$\mathsf{Adv}^{\mathrm{comp}}_{\mathcal{A},\mathrm{DAS},f}(\lambda) = \Pr\left[\mathbf{G}^{comp}_{\mathcal{A},\mathrm{DAS},f}(\lambda) = 1\right],$$

where the game $\mathbf{G}^{\mathrm{comp}}_{\mathcal{A},\mathrm{DAS},f}$ is defined in Fig. 4.

Remark 1. Note that our Repair protocol takes as input an index $i \in [n]$ and tries to repair the ith chunk. But, there exist encoding schemes which only support a global repair algorithm, where the chunks can only be repaired in a particular order. This can be easily modeled by defining a global repair protocol. In the rest of the paper, we will focus on schemes that support local repair for simplicity.

2.2 Security requirements

We define two notions of soundness, which we call the optimistic-case and the worst-case soundness.

Game $\mathbf{G}^{\text{comp}}_{\mathcal{A},\text{DAS},f}$ with respect to an adversary $\mathcal{A}$.

1 : $\text{pp} \leftarrow\!\$\ \text{Setup}(1^\lambda)$
2 : $(n, \text{data}, \text{st}) \leftarrow\!\$\ \mathcal{A}(\lambda, \text{pp})$
3 : $(\text{com}, d_1, \ldots, d_n) \leftarrow \text{Disperse}(\text{data}, n)$
4 : / $\mathcal{A}$ returns the set of corrupt nodes $C \subseteq [n]$ and oracle access to them
5 : $(C, \{O_i(\cdot)\}_{i\in C}) \leftarrow\!\$\ \mathcal{A}(\text{com}, \{d_i\}_{i\in[n]}, \text{st})$
6 : $s \leftarrow\!\$\ \text{Verify}(n, \text{com})\langle \{O_i(\text{st}, \cdot)\}_{i\in C}, \{\text{N}_i(d_i, \cdot)\}_{i\in[n]\setminus C}\rangle$
7 : **return** $|C| \le fn \wedge s = 0$

Fig. 4. The completeness game $\mathbf{G}^{\text{comp}}_{\mathcal{A},\text{DAS},f}$ for a Data availability sampling scheme DAS. It is parametrized with respect to f, denoting the fraction of corrupt storage nodes.

Game $\mathbf{G}^{\text{sound-worst}}_{\mathcal{A},\text{DAS},\delta}$ with respect to an adversary $\mathcal{A}$.

1 : $\text{pp} \leftarrow\!\$\ \text{Setup}(1^\lambda)$
2 : $(n, \text{com}, C, \{O_i(\cdot)\}_{i\in C}, \text{st}) \leftarrow\!\$\ \mathcal{A}(\lambda, \text{pp})$ / $\ell = \ell(n,\lambda), C \subseteq [n]$
3 : $\forall j \in [\ell], (s_j, t_j) \leftarrow\!\$\ \text{Verify}(\text{com})\langle \{O_i(\text{st}, j, \cdot)\}_{i\in C}\rangle$
4 : $s = \Sigma_{j\in[\ell]} s_j$ / added as integers
5 : $\text{GoodSamp} := (s \ge \delta\ell)$ / Many verifiors are convinced
6 : $d \leftarrow \text{Ext}(\{t_j\}_{j\in[\ell]})$
7 : **return** $\text{GoodSamp} \wedge (d = \perp \vee \text{Commit}(d) \neq \text{com})$

Fig. 5. The worst-case soundness game $\mathbf{G}^{\text{sound-worst}}_{\mathcal{A},\text{DAS},\delta}$ for a Data availability sampling scheme DAS. It is parametrized with respect to δ, the fraction of verifiors that the adversary needs to convince. Note that we consider a strong adversary, that can (i) link which queries came from which verifior – this is formalized by giving the verifior index j as input to the adversary's oracles O_i in ln. 3, and (ii) adaptively choose which queries to answer after seeing all of them.

Worst-case Soundness. A DAS scheme is said to achieve worst-case soundness if it is possible to extract the committed data if enough verifiors are convinced that the data is available. More formally, we consider a scenario where the adversary controls the dispersor as well as an arbitrarily large fraction of storage nodes. Even then, if the adversary can convince a subset of $\delta\ell$ of the ℓ verifiors that the data is available, then the Ext algorithm must be able to successfully recover the full data from the sampling transcripts. As mentioned in Sec. 1.1, this notion generalizes the extractor-based definition in [31] to our model. The formal definition can be found in Def. 2.2.

Definition 2.2. We say that a data availability sampling scheme DAS is δ-worst-case sound if for all PPT adversaries $\mathcal{A}$, the following probability is negligible in λ:

$$\text{Adv}^{\text{sound}-\text{worst}}_{\mathcal{A},\text{DAS},\delta}(\lambda) = \Pr\left[\mathbf{G}^{\text{sound-worst}}_{\mathcal{A},\text{DAS},\delta}(\lambda) = 1\right]$$

where $\mathbf{G}^{\text{sound-worst}}_{\mathcal{A},\text{DAS},\delta}$ is as defined in Fig. 5 and the probability is defined over the randomness of the adversary, and the random coins of the Verify protocol.

We remark that this is also similar to a notion called global safety defined in [27].

Optimistic-case Soundness. This is a new definition wherein we consider a weaker adversary, which controls the dispersor, but only a small fraction of the storage nodes. We say that a DAS scheme achieves optimistic-case soundness

Game $\mathbf{G}^{\text{sound-opt}}_{\mathcal{A},\text{DAS},f,\delta}$ with respect to an adversary $\mathcal{A}$.

1 : $\mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda)$
2 : / $\mathcal{A}$ returns n, com, $C \subseteq [n]$, oracles for corrupt nodes, chunks $\{d_i\}_{i\in[n]\setminus C}$, some of which may be $\perp$.
3 : $(n, \mathsf{com}, C, \{d_i\}_{i\in[n]\setminus C}, \{O_i(\cdot)\}_{i\in C}, \mathsf{st}) \leftarrow\!\$\ \mathcal{A}(\lambda, \mathsf{pp})$
4 : $\forall\ j \in [\ell]$:
5 : $\quad (s_j, t_j) \leftarrow\!\$\ \mathsf{Verify}(\mathsf{com})\langle \{O_i(\mathsf{st}, j, \cdot)\}_{i\in C}, \{\mathsf{N}_i(d_i, \cdot)\}_{i\in[n]\setminus C}\rangle$
6 : $d \leftarrow\!\$\ \mathsf{Retrieve}(\mathsf{com})\langle \{O_i(\mathsf{st}, \cdot)\}_{i\in C}, \{\mathsf{N}_i(d_i, \cdot)\}_{i\in[n]\setminus C}\rangle$
7 : $s = \Sigma_{j\in[\ell]} s_j$ / added as integers
8 : $\mathsf{GoodSamp} := (s \geq \delta\ell)$
9 : **return** $\mathsf{GoodSamp} \wedge |C| \leq fn \wedge (d = \perp \vee \mathsf{Commit}(d) \neq \mathsf{com})$

Fig. 6. The optimistic-case soundness game $\mathbf{G}^{\text{sound-opt}}_{\mathcal{A},\text{DAS},f,\delta}$ for a Data availability sampling scheme DAS. It is parametrized with respect to f, the fraction of corrupt nodes, and δ, denoting the fraction of verifiors that need to succeed in order for sampling to be successful. Note that the adversary can (i) link which queries came from which verifiors, (ii) adaptively choose which queries to answer after seeing all of them. We consider static corruptions here for simplicity, but in our analysis, we also allow the adversary to adaptively corrupt storage nodes in the system.

if no such adversary can convince more than δ fraction of verifiors and make the retrieval protocol fail. In other words, a DAS scheme with optimistic-case soundness guarantees that retrieval will succeed as long as the adversary is not too powerful. This is formalized in Fig. 6 and Def. 2.3.

Repair Safety. We define an additional notion of soundness, which we call repair safety. Herein, we again consider an adversary that controls the dispersor and a small fraction of the storage nodes. A DAS scheme is said to achieve repair safety if it is possible to locally repair all the missing chunks if many verifiors are convinced that the data is available. Note that we do not mandate DAS schemes to satisfy this property, so as to avoid overly restricting the design space. Instead, repair safety can be viewed as a "nice-to-have" property that may be targeted in specific applications. We include a formal definition of repair safety in App. C.

Definition 2.3. We say that a data availability sampling scheme DAS is (δ, f)-optimistic-case sound if for all PPT adversaries $\mathcal{A}$, the following probability is negligible in λ:

$$\mathsf{Adv}^{\text{sound}-\text{opt}}_{\mathcal{A},\text{DAS},f,\delta}(\lambda) = \Pr\left[\mathbf{G}^{\text{sound-opt}}_{\mathcal{A},\text{DAS},f,\delta}(\lambda) = 1\right]$$

where the probability is defined over the randomness of the adversary, the random coins of the Verify and Retrieve protocols.

Remark 2 (Comparison with [31])**.** Our definition extends the definition from [31] to capture several important aspects:

- We explicitly model the protocols for Disperse, Repair and Retrieve in our syntax.
- We strengthen their extractor-based definition. Specifically, we allow the adversary to link the queries – it knows which queries are made by which verifior. This can be seen in ln. 3 of the worst-case soundness game in Fig. 5. We also account for the fact that there may be a large number of verifiors, out of which, the adversary can choose which subset of verifiors' queries it answers. This can be seen in lns. 3 to 5 of the same game.

- The notion of optimistic-case soundness captures the requirement that retrieval should work if the adversary is not too powerful – specifically, it only corrupts up to f fraction of the storage nodes. Here again, the adversary is allowed to know which queries were made by which verifior.

Remark 3 (Comparison with Vote-based Data Availability (VDA))**.** We briefly discussed the main differences between VDA and DAS in Sec. 1, we point out more details here.

- As mentioned in Sec. 1, VDA can be formalized with essentially the same definition. The only difference is that the verification process in DAS is interactive, wherein the verifior queries the storage nodes to check if the data is indeed available. This is in contrast to VDA, where the Verify protocol is non-interactive, by simply verifying the signatures given by the storage nodes.
- An additional benefit of the interactive verification is that it can be used in permission-less systems. This is in contrast to VDA systems, where the verifiors need to know the public keys of the storage nodes, to verify the signatures.

2.3 Efficiency requirements

In addition to the security requirements defined above, it is desirable to have low computational and communication complexity for all the protocols. We define the performance metrics which will be used to compare different DAS constructions below:

- **Node storage**. This refers to the amount of data that each storage node has to store, i.e. the size of each chunk d_i. We require this to be smaller than the full data size. This rules out a trivial dispersal algorithm where each node stores the full codeword.
- **Local repair complexity**. For local repair, we define two metrics. The first is the total bandwidth usage for repairing a single chunk, which we require to be smaller than the full data size. This rules out DAS constructions such as [30, 31] and even Ethereum's Fulu DAS proposal, where the only way to repair is to essentially run the full Retrieve algorithm. The second is the number of gossip subnets or topics that a node needs to connect to, in order to repair its chunk. Note that, as mentioned above, we focus on repair during dispersal, in scenarios where the chunk was never dispersed by the dispersor. Moreover, for sake of comparison, we assume that chunks are dispersed over the network via a peer-to-peer gossip protocol (see Sec. 3.3 for more details).
- **Dispersor work**. This refers to the computation complexity of the Disperse algorithm. This usually involves encoding and committing to the payload.
- **Sampling bandwidth**. This refers to the bandwidth usage of a single verifior. This can be computed based on the number of samples each verifior should query, so as to achieve a certain level of worst-case soundness as well as optimistic-case soundness for a particular amount of corruption f.

3 Data Availability Sampling from Codes and Commitments

In this section, we describe our generic framework to build a secure DAS scheme from a locally correctable code and a corresponding erasure code commitment. We start with formally defining both the building blocks, and then we present the framework in Sec. 3.3.

3.1 Locally Correctable Codes

As mentioned in Sec. 1.1, we must use locally correctable codes (LCCs) to achieve efficient repair. We now present a formal definition.

Definition 3.1. A (k, M, Δ, r)–locally-correctable code $\mathsf{C} : \Sigma^k \to \Gamma^M$ is a tuple of the following PPT algorithms

- $\mathsf{Enc}(m) \to \Pi$ is a deterministic algorithm that takes as input a message $m \in \Sigma^k$ and outputs its encoding $\Pi \in \Gamma^M$. The values k and M are referred to as the message length and the codeword length respectively. We will use $\mathsf{C}(\Sigma^k) \subseteq \Gamma^M$ to denote the space of all codewords.
- $\mathsf{Dec}(\hat{\Pi}) \to m$ is a deterministic algorithm that takes as input an encoding with erasures (and *not errors*) $\hat{\Pi}$ and outputs the decoded message m or $\perp$. Moreover, if $\hat{\Pi}$ contains at least $(1-\Delta)M$ symbols of the encoding, then decoding is guaranteed to succeed.
- $\mathsf{LocalCorrect}(i, \hat{\Pi}) \to \Pi_i$ is a randomized local correction algorithm, that, for any $i \in [M]$, given access to an encoding $\hat{\Pi}$ with erasures, outputs the ith symbol of the encoding Π_i or $\perp$. This algorithm is parametrized by r, also referred to as the query complexity, which denotes the number of symbols of the encoding that are used to repair any symbol. We have that $r \leq (1-\Delta)M$, since a trivial local correction algorithm can just run the decoding algorithm using any $1-\Delta$ fraction of the codeword symbols.

We may use $\mathsf{LocalCorrect}(i, \hat{\Pi}; \zeta)$ to denote the deterministic version of the local correction algorithm, which takes randomness $\zeta \leftarrow\!\$\ \{0,1\}^\lambda$ as input.

Here, Δ is referred to as the minimum distance of the code.

The rate of this code is defined as $\frac{k \cdot \log(|\Sigma|)}{M \cdot \log(|\Gamma|)}$. This represents the ratio of the size of the message and its encoding.

Systematic codes. We say that a code C has a systematic encoding if the message m is contained in the codeword $\mathsf{Enc}(m)$. Such a systematic encoding makes it easy to retrieve the message from the codeword – this will allow for an efficient Retrieve algorithm for our data availability scheme. We formally define systematic codes in Def. 3.2 below, similar to the one in [31].

Definition 3.2. A $(k, M, \Delta, r, \delta, \epsilon)$–locally-correctable code $\mathsf{C} : \Sigma^k \to \Gamma^M$ is systematic if there exist deterministic polynomial-time algorithms $\mathsf{Find} : [k] \to [M]$ and $\mathsf{Proj} : [k] \times \Gamma \to \Sigma$ such that:

- For all messages $m \in \Sigma^k$ and $i \in [k]$, let $\hat{m} = \mathsf{Enc}(m)$, and $\hat{i} = \mathsf{Find}(i)$. Then, $\mathsf{Proj}(i, \hat{m}_{\hat{i}}) = m_i$.
- Let $I \subseteq [M]$ such that $|I| \geq \Delta \cdot M$, and let $\{\hat{m}_i\}_{i \in I} \in \Gamma^{|I|}$ denote an encoding with erasures. Let $m = \mathsf{Dec}(\{\hat{m}_i\}_{i \in I})$. Then, for any $i \in [k]$ such that $\mathsf{Find}(i) \in I$, $\mathsf{Proj}(i, \hat{m}) = m_i$.

3.2 Locally-Correctable EC Commitments

Erasure-code (EC) commitments, introduced by [31] are a generalization of vector commitments and polynomial commitments. At a high level, an erasure code commitment with respect to a code C allows committing to a message, such that the encoding of this message (with respect to C) can be opened at any position. Moreover, they satisfy a stronger notion of binding called code-binding, which says that, any set of openings produced by a computationally bounded adversary is guaranteed to be consistent with atleast one codeword of the erasure code.

We define a special type of commitment, called *locally correctable erasure code commitments*, which, in addition to being a secure *erasure code commitment*, also have a notion of local correction. Informally, if it is possible to repair the codeword symbol for a position using a set of positions of the codeword, then it is also possible to repair the opening proof for this position using the proofs for the same set of positions. We now give a formal definition.

Syntax. Consider a (k, M, Δ, r)–locally correctable erasure code $\mathsf{C} : \Sigma^k \to \Gamma^M$. A locally correctable erasure code commitment scheme Com for C is a tuple of five PPT algorithms $(\mathsf{CSetup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{CVerify}, \mathsf{CLocalCorrect})$ with the following syntax:

- $\mathsf{CSetup}(1^\lambda) \to \mathsf{ck}$ takes as input the security parameter and outputs a commitment key ck.
- $\mathsf{Commit}(\mathsf{ck}, m) \to \mathsf{com}$ takes as input a commitment key and a message $m \in \Sigma^k$, and outputs a commitment com.
- $\mathsf{Open}(\mathsf{ck}, m, i) \to \pi$ takes as input commitment key, the message m, an index $i \in [M]$, and outputs an opening π_i.
- $\mathsf{CVerify}(\mathsf{ck}, \mathsf{com}, i, \hat{m}_i, \pi) \to b$ is a deterministic algorithm, that takes as input a commitment key ck, a commitment com, an index i, a claimed opening $\hat{m}_i \in \Gamma$ along with a proof π_i for this index, and outputs a bit $b \in \{0, 1\}$.
- $\mathsf{CLocalCorrect}(\mathsf{ck}, \mathsf{com}, i, \hat{\Pi}, \{\pi_j\}_{j \in \hat{\Pi}}) \to \pi_i$ is a randomized algorithm, which takes as input the commitment key, the commitment, an index i, an encoding with erasures $\hat{\Pi}$, along with the opening proofs π_j for all the positions present in the encoding, and outputs the opening proof for position i or $\perp$. We will use $\mathsf{CLocalCorrect}(\mathsf{ck}, \mathsf{com}, i, \hat{\Pi}, \{\pi_j\}_{j \in \hat{\Pi}}; \zeta)$ to denote the deterministic version which takes randomness $\zeta \leftarrow\!\$\ \{0, 1\}^\lambda$ as input.

To allow for efficient DAS constructions, we also define another algorithm called $\mathsf{BatchOpen}(\mathsf{ck}, m) \to \{\pi_i\}_{i \in [M]}$, which takes as input the commitment key and the state, and outputs the opening proofs for all positions $i \in [M]$. As we will see in Sec. 3.3, commitment schemes that support an efficient batch opening algorithm help reduce the computation load on the dispersor in DAS constructions.

Note that we do not need the commitment scheme to be hiding, because the committed data in DAS schemes does not need to be secret. Hence, we restrict the Commit and Open algorithms to be deterministic.

We now formally define all the properties of a locally-correctable erasure code commitment.

Correctness. The following definition captures correctness, which says that honestly generated opening proofs must verify successfully for all messages and all indices.

Definition 3.3. An erasure code commitment scheme Com for a code C is said to be η-correct, if, for all $\lambda \in \mathbb{N}$, for every ck in the support of $\mathsf{CSetup}(1^\lambda)$, every $m \in \Sigma^k$ and every index $i \in [M]$, the following probability is bounded by $\eta(\lambda)$:

$$\Pr\left[\mathsf{CVerify}(\mathsf{ck}, \mathsf{com}, i, \hat{m}_i, \pi_i) = 0 \;\middle|\; \begin{array}{c} \mathsf{com} \leftarrow\!\$\ \mathsf{Commit}(\mathsf{ck}, m) \\ \hat{m} \leftarrow \mathsf{C.Enc}(m) \\ \pi_i \leftarrow\!\$\ \mathsf{Open}(\mathsf{ck}, m, i) \end{array}\right]$$

where the probability is taken over the randomness of Commit and Open algorithms.

For commitment schemes supporting batch opening, we also define correctness for BatchOpen, which says that, all of the proofs generated by BatchOpen must verify, if everything is run honestly.

Definition 3.4. An erasure code commitment scheme Com for a code C is said to be η_b-correct with respect to batch openings, if, for all $\lambda \in \mathbb{N}$, for every $\mathsf{ck} \leftarrow\!\$\ \mathsf{CSetup}(1^\lambda)$, and every $m \in \Sigma^k$, the following probability is bounded by $\eta_b(\lambda)$:

$$\Pr\left[\begin{array}{c} \exists\, i \in [M] \text{ s.t.} \\ \mathsf{CVerify}\left(\begin{array}{c} \mathsf{ck}, \mathsf{com}, \\ i, \hat{m}_i, \pi_i \end{array}\right) = 0 \end{array} \;\middle|\; \begin{array}{c} \mathsf{com} \leftarrow\!\$\ \mathsf{Commit}(\mathsf{ck}, m) \\ \hat{m} \leftarrow \mathsf{C.Enc}(m) \\ \{\pi_i\}_{i \in [M]} \leftarrow\!\$\ \mathsf{BatchOpen}(\mathsf{ck}, m) \end{array}\right]$$

where the probability is taken over the randomness of Commit and Open algorithms.

Binding. We define four notions of binding.

- *Position-binding.* A commitment scheme is position-binding if the adversary cannot open the commitment to two different values at the same position.

  Definition 3.5. An erasure code commitment scheme Com for a code C is said to be position-binding if for all $\lambda \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$, the following advantage is negligible in λ:

$$\mathsf{Adv}^{\text{pos-bind}}_{\text{Com},\mathsf{C},\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c} \mathsf{CVerify}(\mathsf{ck},\mathsf{com},i,\hat{m},\pi)=1 \wedge \\ \mathsf{CVerify}(\mathsf{ck},\mathsf{com},i,\hat{m}',\pi')=1 \wedge \\ \hat{m}\neq\hat{m}' \end{array} \middle| \begin{array}{c} \mathsf{ck}\leftarrow\!\!\$\ \mathsf{CSetup}(1^\lambda) \\ \left(\begin{array}{c}\mathsf{com},i,\\ \hat{m},\pi,\\ \hat{m}',\pi'\end{array}\right)\leftarrow\!\!\$\ \mathcal{A}(\mathsf{ck}) \end{array}\right]$$

- *Code-binding.* Code-binding means that the adversary cannot open an erasure code commitment on a set of positions so that no codeword matches those openings.

  Definition 3.6. An erasure code commitment scheme Com for a code C is said to be code-binding if for all $\lambda \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$, the following advantage is negligible in λ:

$$\mathsf{Adv}^{\text{code-bind}}_{\text{Com},\mathsf{C},\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c} \forall i\in J: \\ \mathsf{CVerify}(\mathsf{ck},\mathsf{com},i,\hat{m}_i,\pi_i)=1 \\ \wedge\neg\left(\begin{array}{c}\exists\, c\in\mathsf{C}(\Sigma^k):\\ \forall\, i\in J, c_i=\hat{m}_i\end{array}\right) \end{array} \middle| \begin{array}{c} \mathsf{ck}\leftarrow\!\!\$\ \mathsf{CSetup}(1^\lambda) \\ \left(\begin{array}{c}\mathsf{com},J,\\ \{\hat{m}_i,\pi_i\}_{i\in J}\end{array}\right)\leftarrow\!\!\$\ \mathcal{A}(\mathsf{ck}) \end{array}\right]$$

- *Reconstruction-binding.* This property requires that the adversary cannot provide two sets of openings for the same commitment, such that reconstructing from these sets leads to inconsistent messages.

  Definition 3.7. An erasure code commitment scheme Com for a code C is said to be reconstruction-binding if for all $\lambda \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$, the following advantage is negligible in λ:

$$\mathsf{Adv}^{\text{rec-bind}}_{\text{Com},\mathsf{C},\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c} \forall i\in J:\mathsf{CVerify}(\mathsf{ck},\mathsf{com},i,\hat{m}_i,\pi_i)=1 \wedge \\ \forall i\in J':\mathsf{CVerify}(\mathsf{ck},\mathsf{com},i,\hat{m}'_i,\pi'_i)=1 \wedge \\ |J|\geq(1-\Delta)M\wedge|J'|\geq(1-\Delta)M\wedge \\ \perp\notin\{m,m'\}\wedge m\neq m' \end{array} \middle| \begin{array}{c} \mathsf{ck}\leftarrow\!\!\$\ \mathsf{CSetup}(1^\lambda) \\ \left(\begin{array}{c}\mathsf{com},J,J'\\ \{\hat{m}_i,\pi_i\}_{i\in J},\\ \{\hat{m}'_i,\pi'_i\}_{i\in J'}\end{array}\right)\leftarrow\!\!\$\ \mathcal{A}(\mathsf{ck}) \\ m\leftarrow\mathsf{Dec}(\{\hat{m}_i\}_{i\in J}) \\ m'\leftarrow\mathsf{Dec}(\{\hat{m}'_i\}_{i\in J'}) \end{array}\right]$$

- *Extractability.* A commitment scheme is extractable if there is an efficient algorithm CExt that can extract the committed message m from any commitment com output by an adversary, as long as the adversary provides at least one opening. Moreover, when committing to the extracted message m, one obtains the same commitment com. Def. 3.8 formalizes this notion. Satisfying extractability typically requires using the Algebraic Group Model [28, 31].

  Definition 3.8. An erasure code commitment scheme Com for a code C is said to be extractable if there exists a PPT algorithm CExt such that, for all $\lambda \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$, the following advantage is negligible in λ:

$$\mathsf{Adv}^{\text{extr}}_{\text{Com},\mathsf{C},\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c} \mathsf{CVerify}(\mathsf{ck},\mathsf{com},i,\hat{m},\pi)=1\wedge \\ \mathsf{Commit}(\mathsf{ck},m)\neq\mathsf{com} \end{array} \middle| \begin{array}{c} \mathsf{ck}\leftarrow\!\!\$\ \mathsf{CSetup}(1^\lambda) \\ (\mathsf{com},i,\hat{m},\pi)\leftarrow\!\!\$\ \mathcal{A}(\mathsf{ck}) \\ m\leftarrow\mathsf{CExt}(\mathsf{ck},\mathsf{com},i,\hat{m},\pi) \end{array}\right]$$

These four notions of binding help simplify the security analysis of our DAS compiler in Sec. 3.3. We refer the reader to [31] for a detailed overview of different binding notions and their relations.

Local Correction for Openings. Def. 3.9 formalizes the notion of local correction openings.

Definition 3.9. An erasure code commitment scheme Com for a code C is said to be locally correctable for openings, if for all PPT adversaries $\mathcal{A}$, the following is negligible in λ:

$$\mathsf{Adv}^{\mathrm{loc-cor-open}}_{\mathcal{A},\mathsf{Com}}(\lambda) = \Pr\left[\begin{array}{l|l} \mathsf{C.LocalCorrect}(i,\{\hat{m}_j\}_{j\in J};r) \neq \bot \wedge & \mathsf{ck} \leftarrow\!\!\$\ \mathsf{CSetup}(1^\lambda) \\ \forall j \in J : \mathsf{CVerify}(\mathsf{ck},\mathsf{com},j,\hat{m}_j,\pi_j) = 1 \wedge & \left(\begin{array}{c}\mathsf{com}, i, J, \zeta,\\ \{\hat{m}_j,\pi_j\}_{j\in J}\end{array}\right) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{ck}) \\ \mathsf{CLocalCorrect}\left(\begin{array}{c}\mathsf{ck},\mathsf{com},i,\\ \{\hat{m}_j,\pi_j\}_{j\in J};\zeta\end{array}\right) = \bot & \end{array}\right]$$

3.3 Data Availability from Codes and Commitments

We are now ready to describe our generic framework that allows constructing DAS scheme from a locally correctable code C and a corresponding erasure code commitment Com.

The network model. While our compiler can be realized with any network model, we do make two assumptions about its behavior:

- If a sampling client successfully receives a particular "chunk" of the encoding, then all the honest storage nodes, which are supposed to store this chunk, must also have it. This can be realized using many different network models, for example (i) by using a peer-to-peer gossip protocol, wherein the dispersor disperses each chunk of the encoding on a different gossip topic or subnet. Any storage node or sampling client can simply subscribe to this subnet to get this chunk. Or (ii) by requiring the sampling clients to share the received chunk with the storage nodes who need to store this chunk.
- If any honest storage node is storing a particular chunk of the encoding, then it is possible to retrieve this chunk from the network at any time. This again can be realized in many ways, for example, maintaining a peer-to-peer gossip protocol as described above, or, by keeping track of which storage node is storing what chunk on a public bulletin-board, so that anyone can directly query the corresponding nodes for any chunk.

We now present a high-level overview before formally describing the compiler. Note that our description is agnostic to what network model is used.

- *Setup.* The setup simply runs the CSetup algorithm of the erasure code commitment scheme.
- *Dispersal.* The dispersor first applies the locally correctable code to the data to obtain a codeword Π. Then it computes a commitment com to the data using the erasure code commitment, along with an opening proof for each index of the codeword. This is where the BatchOpen algorithm helps – it allows the dispersor to compute all the opening proofs efficiently. Then, the codeword and the opening proofs are dispersed on the network. Node i computes $j_i = i \mod M$ where M is the length of the codeword, and receives the j_ith symbol of the codeword from the network, along with an opening proof for this symbol with respect to com. Note that, we assume that n is a multiple of M for simplicity.
- *Verification.* The verifior client samples indices in $[M]$ uniformly at random. For each index, it queries the network for this index of the codeword. Once the verifior receives the codeword symbol for this index, along with an opening proof, it checks if the proof is valid with respect to the commitment. It outputs one only if the proofs for all the queried samples are valid. We note here that the number of samples that each verifior queries is a parameter of the scheme, which can be tuned based on the desired security vs efficiency tradeoff.

Setup(1^λ):

(1) ck $\leftarrow\!\!\$$ Com.CSetup(1^λ). Output pp $\leftarrow$ ck.

Disperse(data, n):

(1) Compute $\Pi \leftarrow$ C.Enc(data) and com $\leftarrow$ Com.Commit(ck, data).
(2) Compute $\{\pi_j\}_{j\in[M]} \leftarrow$ Com.BatchOpen(ck, m). / Compute opening proofs for all positions
(3) For each $i \in [n]$, let $j_i := i \mod M$. Set $d_i = (\Pi_{j_i}, \pi_{j_i})$.
(4) Output $\{d_1, \ldots, d_n\}$, com.

Verify(com):

(1) Let $s = 1$, and initialize a list $t = \perp$. For all $i \in [Q]$: / $Q = Q(n, \lambda)$
   (a) Sample $j_i \leftarrow\!\!\$ [M]$.
   (b) Retrieve $(\hat{m}_{j_i}, \pi_{j_i})$ from the network.
   (c) If $\hat{m}_{j_i} \neq \perp$, set $t \leftarrow t \cup (j_i, \hat{m}_{j_i}, \pi_{j_i})$. If Com.CVerify(ck, com, $j_i, \hat{m}_{j_i}, \pi_{j_i}$) $= 0$, then set $s \leftarrow 0$.
(2) Output (s, t).

Ext(com, $\mathsf{tran}_1, \ldots, \mathsf{tran}_\ell$): / $\ell = \ell(n, \lambda)$

(1) Parse tran_i as $\{(j_{i,1}, \hat{m}_{j_{i,1}}, \pi_{j_{i,1}}), \ldots, (j_{i,Q}, \hat{m}_{j_{i,Q}}, \pi_{j_{i,Q}})\}$ for all $i \in [\ell]$.
(2) Let $S = \perp, T = \perp$. For each $i \in [\ell], k \in [Q]$, if Com.CVerify(ck, com, $j_{i,k}, \hat{m}_{j_{i,k}}, \pi_{j_{i,k}}$) $= 1$, then set $S \leftarrow S \cup (j_{i,k}, \hat{m}_{j_{i,k}}, \pi_{j_{i,k}})$ and $T \leftarrow T \cup \{j_{i,k}\}$.
(3) If S contains two tuples for the same position, i.e. $(j, \hat{m}_j, \pi_j) \in S$ and $(j, \hat{m}'_j, \pi'_j) \in S$, such that $\hat{m}_j \neq \hat{m}'_j$, then output $\perp$.
(4) Otherwise, output C.Dec($\{\hat{m}_j\}_{j\in T}$).

Fig. 7. The Setup, Disperse, Verify and Ext algorithms for our general compiler to construct a DAS[C, Com] scheme from a locally correctable code C and a corresponding erasure code commitment Com.

- *Extraction.* The Ext algorithm collects all the codeword symbols from the transcripts whose opening proofs are valid. It then simply runs the Dec algorithm for the locally correctable code.
- *Retrieval.* The Retrieve protocol tries to run the Dec algorithm of the code C. Specifically, it tries to collect all the symbols of the encoding, by querying its peers over the network. Once it receives enough symbols, it can run the Dec algorithm for the locally correctable code. If the code C is systematic, the retrieval protocol can directly try to retrieve the symbols corresponding to codeword positions Find(i) for all $i \in [k]$ – this allows for a more efficient retrieval when most of the symbols of the codeword are indeed available.
- *Repair.* To repair the ith chunk of the encoding, the node simply runs the LocalCorrect algorithm of the code C. For each index of the codeword needed by the algorithm, the node queries its peers over the network (e.g. by subscribing to subnets in the gossip network model). Lastly, to repair the opening proof for this chunk, it runs the CLocalCorrect algorithm of the commitment scheme Com.

Figs. 7 and 8 present the formal description of our compiler.

Completeness. The completeness of the DAS scheme follows from the completeness of the commitment scheme.

Theorem 3.10. *For every adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathrm{comp}}_{\mathcal{A},\mathsf{DAS}[\mathsf{C},\mathsf{Com}_\mathsf{C}]}(\lambda) \leq negl(\lambda).$$

The proof can be found in App. B.

Retrieve(com):

(1) If the code is systematic, then, for all $i \in [k]$, query the network for the position $j_i = \mathsf{Find}(i)$ to get $(\hat{m}_{j_i}, \pi_{j_i})$.
(2) If $(\hat{m}_{j_i}, \pi_{j_i})$ is retrieved and $\mathsf{CVerify}(\mathsf{ck}, \mathsf{com}, j_i, \hat{m}_{j_i}, \pi_{j_i}) = 1$ for j_i for all $i \in [k]$, return $\{\mathsf{Proj}(i, \hat{m}_{j_i})\}_{i\in[k]}$.
(3) Otherwise, let $i = 1, S = \perp, T = \perp$.
   (a) Query the network for position i. Let $(\hat{m}_i, \pi_i)$ denote the response.
   (b) If $\hat{m}_i \neq \perp$ and $\mathsf{CVerify}(\mathsf{ck}, \mathsf{com}, i, \hat{m}_i, \pi_i) = 1$, then set $T \leftarrow T \cup \{i\}$, $S \leftarrow S \cup (i, \hat{m}_i, \pi_i)$.
   (c) If $|T| \geq (1 - \Delta)M$ then go to Item 4. Otherwise, if $i = M$ then output $\perp$.
   (d) Otherwise, set $i \leftarrow i + 1$.
(4) If S contains two tuples for the same position, i.e. $(j, \hat{m}_j, \pi_j) \in S$ and $(j, \hat{m}'_j, \pi'_j) \in S$, such that $\hat{m}_j \neq \hat{m}'_j$, then output $\perp$.
(5) Output $\mathsf{C.Dec}(\{\hat{m}_j\}_{j\in T})$.

Repair(com, i):

(1) Run $\hat{m}'_i \leftarrow\!\!\$\ \mathsf{LocalCorrect}(i; \zeta)$ algorithm of the code C for $\zeta \leftarrow\!\!\$\ \{0,1\}^\lambda$. For any position j used by the algorithm, query the network for j to get $(\hat{m}_j, \pi_j)$. Let $S = \perp, T = \perp$. If $\mathsf{CVerify}(\mathsf{ck}, \mathsf{com}, j, \hat{m}_j, \pi_j) = 0$, then set $\hat{m}_j \leftarrow \perp$, otherwise, set $S \leftarrow S \cup \{(j, \hat{m}_j, \pi_j)\}$.
(2) Run $\pi_i \leftarrow\!\!\$\ \mathsf{CLocalCorrect}(\mathsf{ck}, \mathsf{com}, i, \{\hat{m}_j\}_{j\in T}, \{\pi_j\}_{j\in T}; \zeta)$.
(3) Output $(\hat{m}'_i, \pi_i)$.

Fig. 8. The Retrieve and Repair algorithms for our general compiler to construct a DAS[C, Com] scheme from a locally correctable code C and a corresponding erasure code commitment Com.

Worst-case Soundness. Thm. 3.11 below proves δ-worst-case soundness for our DAS construction, assuming that the commitment scheme is binding.

Theorem 3.11. *For every* PPT *adversary* $\mathcal{A}$*, there exist* PPT *adversaries* $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ *and* $\mathcal{B}_4$ *such that,*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{sound-worst}}_{\mathcal{A},\mathsf{DAS[C,Com]}}(\lambda) \leq & \mathsf{Adv}^{\mathsf{pos-bind}}_{\mathcal{B}_1,\mathsf{Com}}(\lambda) + \mathsf{Adv}^{\mathsf{code-bind}}_{\mathcal{B}_2,\mathsf{Com}}(\lambda) + \\ & \mathsf{Adv}^{\mathsf{ext}}_{\mathcal{B}_4,\mathsf{Com}}(\lambda) + \mathsf{Adv}^{\mathsf{rec-bind}}_{\mathcal{B}_3,\mathsf{Com}}(\lambda) + \\ & \binom{\ell}{\delta\ell} \cdot \binom{M}{(1-\Delta)M} \cdot (1-\Delta)^{\delta\ell Q}.\end{aligned}$$

The proof can be found in App. B.

Optimistic-case Soundness. Thm. 3.12 below proves δ-optimistic-case soundness for our DAS construction, assuming that the commitment scheme is binding.

Theorem 3.12. *For any* PPT *adversary* $\mathcal{A}$*, there exist* PPT *adversaries* $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ *and* $\mathcal{B}_4$ *such that:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{sound-opt}}_{\mathcal{A},\mathsf{DAS[C,Com_C]}}(\lambda) \leq & \mathsf{Adv}^{\mathsf{pos-bind}}_{\mathcal{B}_1,\mathsf{Com}}(\lambda) + \mathsf{Adv}^{\mathsf{code-bind}}_{\mathcal{B}_2,\mathsf{Com}}(\lambda) + \\ & \mathsf{Adv}^{\mathsf{ext}}_{\mathcal{B}_4,\mathsf{Com}}(\lambda) + \mathsf{Adv}^{\mathsf{rec-bind}}_{\mathcal{B}_3,\mathsf{Com}}(\lambda) + \\ & \binom{\ell}{\delta\ell} \cdot \binom{M}{(1-\Delta+f)M} \cdot (1-\Delta+f)^{\ell\delta Q}\end{aligned}$$

The proof can be found in App. B.

Efficiency.

- **Node storage.** Each storage node in the system needs to store one symbol of the encoding, a corresponding opening proof and the commitment to the data. More formally, we get, $|\mathsf{com}| + |\pi_i| + \log(|\Gamma|)$.

- **Dispersor work.** The work of the dispersor includes (i) running the Enc algorithm for the code, (ii) committing to the data i.e. running the Commit algorithm, and (iii) computing opening proofs for all positions of the encoding, i.e. the BatchOpen algorithm.
- **Local repair complexity.** To repair a particular symbol of the encoding, one needs r other symbols of the encoding, where r is the query complexity of the underlying code. In a system where a gossip protocol is used to disperse the data, if there is a separate subnet or topic for each chunk i.e. symbol of the encoding, the number of subnets one node has to contact is simply r. Moreover, since we only need to retrieve one symbol and its opening proof from each peer, the total bandwidth usage is r times $\log(|\Gamma|) + |\pi_i|$.
- **Sampling bandwidth** This will be equal to Q times the size of one symbol and its opening proof. The parameter Q can be set so that the advantage of an adversary in the worst-case soundness game and optimistic-case soundness is exponentially small in the statistical security parameter, given an upper bound on the fraction of nodes that the adversary is allowed to corrupt. Specifically, let κ be the statistical security parameter and let f be the fraction of corrupt nodes. We can solve the following equations to get Q:

$$\binom{\ell}{\delta\ell} \cdot \binom{M}{(1-\Delta)M} \cdot (1-\Delta)^{\delta\ell Q} \leq \frac{1}{2^{\kappa}}$$
$$\binom{\ell}{\delta\ell} \cdot \binom{M}{(1-\Delta+f)M} \cdot (1-\Delta+f)^{\delta\ell Q} \leq \frac{1}{2^{\kappa}}$$

Remark 4 (On index sampling)**.** Note that in our Verify algorithm, we sample each index j_i uniformly at random from the space of all indices $[M]$. But, our design and analysis can be easily generalized to other forms of sampling mentioned in [31].

3.4 Proving security of Danksharding

Ethereum's DAS proposal [23] for the upcoming Fulu hard fork can be seen as an instantiation of our generic compiler, hence our analysis can be used to prove soundness of their construction. Specifically, their design uses Reed-Solomon code to encode the data, and KZG commitment as the corresponding erasure-code commitment. Notably, Reed-Solomon codes have no local correction, meaning that to repair even a single symbol, one needs to retrieve data equivalent to the size of the original data. Moreover, the sampling bandwidth analysis in Tab. 1 is based on their specification which states that each verifior samples 8 symbols of the encoding. Unfortunately, as per our soundness analysis, this does not achieve 40 bits of optimistic-case soundness.

4 Our Construction

In this section, we describe how we encode the data and a corresponding erasure code commitment. These can then be combined via the framework given in Sec. 3.3 to get a secure DAS scheme.

4.1 Encoding Design

We use multiplicity code [38, 67] as part of our encoding design, because it provides efficient local correction and good distance. We now present an overview of our code before the formal description.

Encoding. We encode the data with a multiplicity code, which is a generalization of Reed-Solomon and Reed-Muller codes. Specifically, the data is interpreted as a multivariate polynomial h. The encoding then includes evaluations of

this polynomial h along with its derivatives. One symbol (or chunk) of the encoding contains all the evaluations at one point. Informally, these derivatives provide extra redundancy, allowing us to achieve good distance.

The code is parametrized by (i) v, the number of variables in the polynomial used for encoding, (ii) d, the total degree of the polynomial, (iii) m, the order of the multiplicity code, which refers to the number of derivatives that we include in the encoding, and (iv) q, which determines the points over which the polynomial will be evaluated in the encoding. Specifically, the encoding will contain the evaluation of the polynomial h (and its derivatives) at points of the form $(\omega^{i_1}, \ldots, \omega^{i_v})$, for all $i_1, \ldots, i_v \in \{0, \ldots, q-1\}$, where ω is a qth root of unity in F_p.[1] Tab. 3 lists the parameters for our construction. We chose the parameters that provide a reasonable trade-off between local correction and distance of the code.

Systematic encoding. Recall that in Sec. 3.3, we mentioned that it is good to use a systematic encoding since it allows for efficient retrieval when most of the symbols are indeed available on the network. We present the first efficient algorithm to support systematic encoding for multiplicity code, based on [8] (and references therein). For example, if h is a bivariate polynomial with total degree d, then [8] show that we can treat the data as evaluations of h at points of the form (ω^i, ω^j) where $i + j \leq d$. We design an efficient encoding algorithm which can recover the coefficients of h from these evaluations. The key insight we use is that the evaluations of a bi-variate polynomial can be seen as the result of a series of Fast Fourier transformations on the coefficients. Specifically, let $h(X_1, X_2) = \Sigma_{u \in \{0,\ldots,d\}} X_1^u \cdot h_u(X_2)$, where $h_u(X_2)$ is a univariate polynomial with degree bound $d - u$. To obtain evaluations of h from its coefficients, we can first compute the evaluations of $h_u(X_2)$ at all powers of ω using a Fast Fourier transform, for all $u \in \{0, \ldots, d\}$. Then, for any $j \in \{0, \ldots, q-1\}$, we observe that,

$$h(\omega^i, \omega^j) = \Sigma_{u \in \{0,\ldots,d\}} (\omega^i)^u \cdot h_u(\omega^j)$$

for all $i \in \{0, \ldots, q-1\}$. For a given j, we observe that $h(\omega^i, \omega^j)$ in the above equation looks like 'evaluation' of a polynomial with 'coefficients' equal to $h_0(\omega^j), \ldots, h_d(\omega^j)$. Hence, for any $j \in \{0, \ldots, q-1\}$, we can get the evaluation of h at points (ω^i, ω^j) for all i by a Fast Fourier transform (FFT) over the 'coefficients' $h_0(\omega^j), \ldots, h_d(\omega^j)$.

Now, for systematic encoding, we need to obtain all coefficients of h from evaluations. We can do so by simply reversing the above process. At a high level, for each $j \in \{0, q-1\}$, we can obtain the 'coefficients' $h_0(\omega^j), \ldots, h_d(\omega^j)$ from the evaluations $h(\omega^0, \omega^j), \ldots, h(\omega^{q-1}, \omega^j)$ via interpolation. Then, for each $u \in \{0, \ldots, d\}$, we can similarly use the evaluations of $h_u(X_2)$ to obtain its coefficients via interpolation. Note that there are a few subtleties we need to handle because we do not have the evaluations of h at all points, rather only at (ω^i, ω^j) where $i + j \leq d$.

We use an algorithm Interpolate as a subroutine, which, given $d+1$ evaluations of a degree d polynomial at arbitrary points, outputs all coefficients of the polynomial. This can be implemented using the fast interpolation algorithm given in [60, Sec.10.2], which runs in time $O(d \log^2(d))$.

Once we recover the coefficients of h from the data, we use them to compute the evaluations of h and its derivatives at all powers of ω. Fig. 9 describes the full algorithm. For encoding the data with a bivariate polynomial, it takes time $O(d^2 \log^2(d))$ where $d^2 = \Omega(B)$ and B is the length of the data (this follows from the fact that the polynomial must have enough coefficients to encode the data). Moreover, the length of the codeword is q^v, and the alphabet Γ is $\mathsf{F}_p^{\binom{m+v-1}{v}}$.

Decoding. For decoding, let Δ denote the distance of our code. Then, we know that any $(1 - \Delta)$ fraction of the encoded matrix is enough to recover the original data. Specifically, since multiplicity code is linear, we can form a large system of linear equations to solve for the coefficients of the polynomial, which is enough to get the data.

[1] We use roots of unity as evaluation points to allow for efficient algorithms for the commitment, as we will see in Sec. 4.2.

Parameter	Description	Optimal value
v	Number of variables	2
m	Number of derivatives in encoding	2
d	Total degree of polynomial	90
q	Evaluation points in encoding	96

Table 3. Parameters for our encoding design for data of length $B = 4096$. With these parameters, our code achieves distance 0.53 (where as Ethereum Fulu encoding only achieves 0.5). Moreover, for repairing a single chunk, a node needs $d + 1 = 91$ symbols, and in the gossip network model where there is a subnet for each symbol, the number of subnets for local correction is just 91. See Tab. 1 for a full comparison.

Local correction. Let us consider multiplicity coding with bi-variate polynomials for simplicity. Specifically, consider the polynomial $h(X_1, X_2)$ used to encode the data. To repair the chunk of any storage node, we need to repair the evaluation of h and its derivatives at a point $p_j = (\omega^{j_1}, \ldots, \omega^{j_2})$. The key observation we use is that h restricted to any line L of the form $X_2 = aX_1 + b$ is a univariate polynomial $h_L = h(X_1, aX_1 + b)$ of degree d. Hence, to recover the evaluation of h at p_j, it is sufficient to (i) sample a random line passing through p_j, (ii) get the evaluation of h at any $d + 1$ points on this line. This is because, with $d + 1$ evaluations, we can fully recover the univariate polynomial h_L, and hence also the evaluation of h_L at p_j. Similarly, since the derivatives of h also have total degree less than d, we can use the same principle to repair their evaluations.

We note that multiplicity codes support another method of local correction, wherein, for a bi-variate polynomial, we can sample any two lines passing through the point, and then, the evaluations of h and its derivatives at any $\lceil d/2 \rceil$ points on both lines is enough to repair that point (see [38] for a full description). We leave formalizing this repair for our encoding and commitment to future work.

4.2 Commitment Design

We now describe our construction of PSTMult, an erasure code commitment for multiplicity codes. This is the first commitment scheme for multivariate polynomials that supports i) computing batch openings efficiently (also referred to as one-to-many prover in [70, 71]), ii) proving the evaluations of derivatives, as well as iii) local correction of openings. While there are many commitment schemes for multivariate [51, 71] and multilinear [37] polynomials, they do not generalize to our use-case wherein we require all the three properties listed above.

Our commitment is based on the PST commitment scheme [51]. We start with some notation. Let $\mathsf{GroupGen}(1^\lambda)$ denote an algorithm that samples a bilinear group, i.e. $\mathbb{G}_1$ and $\mathbb{G}_2$ with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with $|\mathbb{G}_1| = p$. Let g_1, g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We use additive notation to denote group operations.

In the PST scheme [51], the commitment to a polynomial h is the evaluation of the polynomial at a secret point $\boldsymbol{t}$, computed in the first group, i.e. $[h(\boldsymbol{t})]_1$. For simplicity, let us consider a bivariate polynomial $h(X_1, X_2)$ with degree d (note that our techniques are easily extended to more variables). The opening proof for h at $\boldsymbol{u}$ are commitments to quotient polynomials q_1 and q_2 defined as follows:

$$h(X_1, X_2) = (X_1 - \boldsymbol{u}_1)q_1(X_1, X_2) + (X_2 - \boldsymbol{u}_2)q_2(X_2) + h(\boldsymbol{u}). \tag{1}$$

$\mathsf{Enc}(m \in \mathsf{F}_p^B)$:

(1) Let M and Y be a $(d+1) \times (d+1)$ matrix and F be a $(d+1) \times q$ matrix. For $i \in \{0, d\}$ and $j \in \{0, d-i\}$, $M_{i,j} \leftarrow m_{i(2d+3-i)/2+j}$. / We assume $B \leq \binom{d+2}{2}$
(2) Let $h = \Sigma_{i,j \geq 0, i+j \leq d} h_{i,j} X_1^i X_2^j \in \mathsf{F}_p[X_1, X_2]$ be the polynomial of total degree d.
(3) Let $s = \Sigma_{i \in \{0,\ldots,d\}} s_i X^i \in \mathsf{F}_p^d[X]$ be a univariate polynomial, initialized to zero i.e. $s = 0$. For $c \in \{0, \ldots, d\}$:
   (a) Compute $t_0, \ldots, t_{q-1} \leftarrow \mathsf{FFT}(q, s_0, \ldots, s_d)$.
   (b) $(Y_{0,c}, \ldots, Y_{d-c,c}) \leftarrow \mathsf{Interpolate}(d-c, \{M_{0,c} - t_0, \ldots, M_{d-c,c} - t_{q-1}\})$
   (c) Compute the coefficients $(h_{d-c,0}, \ldots, h_{d-c,c}) \leftarrow \mathsf{Interpolate}(c, Y_{d-c,0}, \ldots, Y_{d-c,c})$
   (d) Set $(F_{d-c,0}, \ldots, F_{d-c,q-1}) \leftarrow \mathsf{FFT}(q, h_{d-c,0}, \ldots, h_{d-c,c})$.
   (e) For $i \in \{d-c, \ldots, d\}$, set $s_i \leftarrow F_{i,c+1}$.
(4) Compute $h'_1(X_1, X_2) = \Sigma_{i,j \geq 0, i+j \leq d} i \cdot h_{i,j} X_1^{i-1} X_2$ and $h'_2(X_1, X_2) = \Sigma_{i,j \geq 0, i+j \leq d} j \cdot h_{i,j} X_1^i X_2^{j-1}$.
(5) Let E_0, E_1, E_2 be $q \times q$ matrices in F_p. For $c \in \{0, \ldots, q-1\}$, set $(E_{0,0,c}, \ldots, E_{0,q-1,c}) \leftarrow \mathsf{FFT}(q, F_{0,c}, \ldots, F_{d,c})$.
(6) For $i \in \{1, 2\}$:
   (a) Write $h'_i(X_1, X_2)$ as $\Sigma_{k \in \{0,\ldots,d\}} X_1^k \cdot h'_{i,(k)}(X_2)$, where $h'_{i,(k)} = \Sigma_{j \in \{0,\ldots,d-k\}} h'_{i,(k),j}$ has degree $\leq d-k$.
   (b) Let F_i be a $(d+1) \times q$ matrix. For $r \in \{0, \ldots, d\}$, compute $(F_{i,r,0}, \ldots, F_{i,r,q-1}) \leftarrow \mathsf{FFT}(q, h'_{i,(r),0}, \ldots, h'_{i,(r),d-r})$.
   (c) For $c \in \{0, \ldots, q-1\}$, compute $(E_{i,0,c}, \ldots, E_{i,q-1,c}) \leftarrow \mathsf{FFT}(q, F_{i,0,c}, \ldots, F_{i,q-1,c})$.
(7) For $x \in \{0, \ldots, q^2 - 1\}$, let $i \leftarrow \lfloor x/q \rfloor$ and $j \leftarrow x \mod q$. Set $\Pi_x \leftarrow (E_{0,i,j}, E_{1,i,j}, E_{2,i,j})$.
(8) Output $\Pi \in (\mathsf{F}_p^3)^{q^2}$.

Fig. 9. The encoding algorithm for our construction. This is the first efficient algorithm that supports systematic encoding for multiplicity codes. We describe the case where data is encoded with a multiplicity code with two variables i.e. $v = 2$, and $m = 2$, i.e. only the first partial derivative with respect to both variables is included in the encoding.

This equation follows from Lem. 4.1, proven in [51]. The verifier can check the proof by checking if the above relation holds at the secret point $\boldsymbol{t}$, via pairings:

$$e\left([h(\boldsymbol{t})]_1 - [y]_1, [1]_2\right) = e\left([q_1(\boldsymbol{t})]_1, [\boldsymbol{t}_1]_2 - [\boldsymbol{u}_1]_2\right) \cdot e\left([q_2(\boldsymbol{t})]_1, [\boldsymbol{t}_2]_2 - [\boldsymbol{u}_2]_2\right)$$

Note that the terms $\{[\boldsymbol{t}_i]_2\}_{i \in [2]}$ and $\{[\boldsymbol{t}_1^i \boldsymbol{t}_2^j]_1\}_{i,j, \in [d]}$ are included in the public parameters (also called a common reference string or CRS), generated via a trusted setup. We now show how we achieve the three properties for this commitment scheme.

Lemma 4.1. *Let v be a positive integer. Consider $\boldsymbol{u} \in \mathsf{F}_p^v$. Let $f \in \mathsf{F}_p[X_1, \ldots, X_v]$ be a v-variate polynomial with total degree d. Then, there exist polynomials $q_i \in \mathsf{F}_p[X_i, \ldots, X_v]$ such that:*

$$f(X_1, \ldots, X_v) = \Sigma_{i \in [v]} (X_i - \boldsymbol{u}_i) \cdot q_i(X_i, \ldots, X_v) + f(\boldsymbol{u})$$

Moreover, these q_i polynomials can be computed in polynomial time. We refer to the $q_1, \ldots, q_v$ polynomials as the quotient polynomials of f with respect to $\boldsymbol{u}$.

4.2.1 Proving evaluations of derivatives. For simplicity, we consider the case where we only need to prove evaluations of the first derivatives, but our techniques generalize to higher-order derivatives as well. Let us use h'_1, h'_2 to denote the first derivatives of h with respect to variables X_1 and X_2 respectively. Let q_1, q_2 be the quotient polynomials for a point $(\boldsymbol{u}_1, \boldsymbol{u}_2)$, as defined in Eq. (1).

LocalCorrect(i):

(1) Let $i_r \leftarrow \lfloor i/q \rfloor$ and $i_c \leftarrow i \mod q$.
(2) Sample a random line $L(\gamma)$ of the form $(a_1\gamma + i_r, a_2\gamma + i_c) \mod p$, passing through (i_r, i_c), such that, there are at least $d+2$ points in $\{0,\ldots,q-1\} \times \{0,\ldots,q-1\}$ that lie on L.
(3) Let $(x_{1,1}, x_{1,2}), \ldots (x_{d+1,1}, x_{d+1,2})$ denote the points on L, except for (i_r, i_c) and let $\gamma_u = (x_{u,1} - i_r)/a_1 \mod p$ for all $u \in [d+1]$.
(4) Query the network for symbol $x_{u,1} \cdot q + x_{u,2}$ for all all $u \in [d+1]$.
(5) Let $(z_{u,0}, z_{u,1}, z_{u,2})$ denote the evaluation at $(x_{u,1}q + x_{u,2})$, for all $u \in [d+1]$.
(6) For $w \in \{0,1,2\}$:
  (a) Compute the univariate polynomial $h^*_w(X) \in \mathsf{F}^d_p$ such that $h^*_w(\gamma_u) = z_{u,w}$ for all $u \in [d+1]$.
  (b) Set $z^*_w \leftarrow h^*_w(0)$.
(7) Output $\hat{\Pi}_i = (z^*_0, z^*_1, z^*_2)$.

Find$(i \in \{0,\ldots,B-1\})$:

(1) Find j_1 such that $(\binom{d+2}{2} - \binom{d+2-j_1}{2}) \leq i < (\binom{d+2}{2} - \binom{d+1-j_1}{2})$.
(2) Compute $j_2 \leftarrow i - (j_1 \cdot (2d+3-j_1)/2)$.
(3) Output $(j_1 \cdot q + j_2)$.

Proj$(\hat{i} \in [B], \hat{\Pi}_{\hat{i}})$:

(1) Parse $\hat{\Pi}_{\hat{i}}$ as (z_0, z_1, z_2).
(2) Output z_0.

Fig. 10. The formal description of local correction , Find and Proj algorithms for our code. We refer the reader to [38, 67] for the decoding algorithm.

The key insight that allows us to prove the evaluations of the derivatives of h is the fact that the evaluation of the quotient polynomial q_i at $\boldsymbol{u}$ is equal to the evaluation of f'_i at $\boldsymbol{u}$. More formally:

$$q_1(\boldsymbol{u}_1, \boldsymbol{u}_2) = f'_1(\boldsymbol{u}),\ q_2(\boldsymbol{u}_2) = f'_2(\boldsymbol{u})$$

This can be seen as the l'Hopital's rule applied over finite fields (we formally prove this in Thm. A.1).

Hence, we can simply prove the evaluations of both the derivatives by opening the quotients q_1 and q_2 at the same point $\boldsymbol{u}$. We compute the quotient polynomials for opening q_1, q_2 by applying Lem. 4.1 again:

$$q_1(X_1, X_2) = (X_1 - \boldsymbol{u}_1)q_{11}(X_1, X_2) + (X_2 - \boldsymbol{u}_2)q_{12}(X_2) + q_1(\boldsymbol{u})$$
$$q_2(X_2) = (X_2 - \boldsymbol{u}_2)q_{22}(X_2) + q_2(\boldsymbol{u}_2)$$

The opening proof for f, f'_1, f'_2 at $\boldsymbol{u}$ can then comprise of commitments to the quotients for f i.e. q_1, q_2, the quotients for q_1 i.e. q_{11}, q_{12} and the quotients for q_2 i.e. q_{22}, i.e. a total of five group elements. The proof can be checked by verifying the openings for f, q_1, q_2 independently via a total of eight pairings.

We now make another observation that allows us to optimize this further. Specifically, the three opening checks can be combined as follows:

$$\begin{aligned} h(X_1,X_2) &= (X_1 - \boldsymbol{u}_1)q_1(X_1, X_2) + (X_2 - \boldsymbol{u}_2)q_2(X_2) + f(\boldsymbol{u}) \\ &= (X_1 - \boldsymbol{u}_1)((X_1 - \boldsymbol{u}_1)q_{11}(X_1, X_2) + (X_2 - \boldsymbol{u}_2)q_{12}(X_2) + q_1(\boldsymbol{u})) \\ &\quad + (X_2 - \boldsymbol{u}_2)((X_2 - \boldsymbol{u}_2)q_{22}(X_2) + q_2(\boldsymbol{u}_2)) + f(\boldsymbol{u}) \end{aligned} \tag{2}$$

$$
\begin{aligned}
&= (X_1 - \boldsymbol{u}_1)^2 q_{11}(X_1, X_2) + (X_1 - \boldsymbol{u}_1)(X_2 - \boldsymbol{u}_2) q_{12}(X_2) \\
&\quad + (X_2 - \boldsymbol{u}_2)^2 q_{22}(X_2) \\
&\quad + (X_1 - \boldsymbol{u}_1) h_1'(\boldsymbol{u}) + (X_2 - \boldsymbol{u}_2) h_2'(\boldsymbol{u}) + h(\boldsymbol{u})
\end{aligned} \tag{3}
$$

We show that it is enough to check the equality in Eq. (3). This means that the prover only needs to provide commitments to q_{11}, q_{12}, q_{22} as the opening, i.e. three group elements. The verifier can then check the proof by checking the above equation via just four pairings. Fig. 11 provides a formal description of this commitment scheme.

4.2.2 Supporting batch openings. Recall that the dispersor needs to compute and disperse the opening proofs for all positions in the codeword. A naive algorithm that just runs the Open algorithm for all the points will take time $O(q^2 \cdot d^2)$, since a single opening requires committing to bivariate polynomials of total degree close to d. We now present a batch opening algorithm, that allows computing all the opening proofs in time $O(q^2 \log(q))$. At a high level, we exploit the fact that the opening proofs for evaluations at roots of unity are highly correlated. Note that this is essentially optimal because even evaluating a bi-variate polynomial at all points (ω^i, ω^j) for $i, j \in \{0, \ldots, q-1\}$ will take time $O(q^2 \log(q))$ (via the algorithm described in Sec. 4.1).

Let us start with some notation. Let ω be the qth root of unity in F_p. For all $i, j \in \{0, \ldots, q-1\}$, we denote by $\pi_{11}^{(i)}, \pi_{12}^{(i,j)}, \pi_{22}^{(i,j)}$ as the three elements in the opening proof for the point (ω^i, ω^j). Specifically, we have

$$\pi_{11}^{(i)} = [q_{11}^{(i)}(\boldsymbol{t})]_1, \pi_{12}^{(i,j)} = [q_{12}^{(i,j)}(\boldsymbol{t})]_1, \pi_{22}^{(i,j)} = [q_{22}^{(i,j)}(\boldsymbol{t})]_1$$

where the quotient polynomials $q_{11}^{(i)}, q_{12}^{(i,j)}$ and $q_{22}^{(i,j)}$ are as defined in Eq. (3) for the point (ω^i, ω^j).

Computing π_{11} for all points. Let us define univariate polynomials $h_0, \ldots, h_d \in \mathsf{F}_p[X_2]$ with degree bounds $d, \ldots, 0$ respectively, such that:

$$h(X_1, X_2) = \Sigma_{u=0}^{d} h_u(X_2) \cdot X_1^u$$

Let us now look at the coefficients of $q_{11}^{(i)}(X_1, X_2)$ polynomial. In Lem. A.3, we show that,

$$
\begin{aligned}
q_{11}^{(i)}(X_1, X_2) &= h_d(X_2) \cdot X_1^{d-2} + \\
&\quad (h_{d-1}(X_2) + 2\omega^i h_d(X_2)) \cdot X_1^{d-3} + \\
&\qquad \vdots \\
&\quad (h_2(X_2) + 2\omega^i h_3(X_2) + \ldots + (d-1)(\omega^i)^{d-2} h_d(X_2)) \cdot X_1^0
\end{aligned}
$$

This can be seen as a generalization of the observation in Feist-Khovratovich [19, Eqn.14] in two ways: (a) handling multivariate polynomials, and (b) computing quotients for higher degree polynomials, i.e. $(X_1 - \omega^i)^2$. We can now re-arrange the terms to get:

$$
\begin{aligned}
&q_{11}^{(i)}(X_1, X_2) = \\
&\quad (h_d(X_2) \cdot X_1^{d-2} + h_{d-1}(X_2) \cdot X_1^{d-3} + \ldots h_2(X_2)) \cdot (\omega^i)^0 + \\
&\quad 2 \cdot (h_d(X_2) \cdot X_1^{d-3} + h_{d-1}(X_2) \cdot X_1^{d-4} + \ldots + h_3(X_2)) \cdot (\omega^i)^1 + \\
&\qquad \vdots \\
&\quad (d-1) \cdot h_d(X_2) \cdot (\omega^i)^{d-2}
\end{aligned} \tag{4}
$$

More succinctly, we can define d polynomials $G_1, \ldots, G_{d-1}$ in $\mathsf{F}_p[X_1, X_2]$ such that:

$$q_{11}^{(i)}(X_1, X_2) = \Sigma_{u \in [d-1]} u \cdot G_u(X_1, X_2) \cdot (\omega^i)^{u-1}$$

Now, we can obtain the opening proofs $\pi_{11}^{(i)}$ for all i in three steps: (i) Compute $w_u \leftarrow [G_u(\tau_1, \tau_2)]_1$ for all u, (ii) Multiply w_u by u, i.e. $w_u \leftarrow u \cdot w_u$, (iii) Run a Discrete Fourier transform on $\{w_1, \ldots, w_{d-1}\}$ to get all the opening proofs $\{\pi_{11}^{(0)}, \ldots, \pi_{11}^{(q-1)}\}$. For step (i), we observe that the polynomials $G_u(X_1, X_2)$ can be expressed as a matrix-vector product:

$$\begin{bmatrix} G_1(X_1, X_2) \\ G_2(X_1, X_2) \\ \vdots \\ G_{d-1}(X_1, X_2) \end{bmatrix} = \begin{bmatrix} h_d(X_2) & h_{d-1}(X_2) & \ldots & h_2(X_2) \\ 0 & h_d(X_2) & \ldots & h_3(X_2) \\ \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & \ldots & h_d(X_2) \end{bmatrix} \cdot \begin{bmatrix} X_1^{d-2} \\ X_1^{d-3} \\ \vdots \\ X_1^0 \end{bmatrix}$$

Therefore, the commitments to $G_1, \ldots, G_{d-1}$ can also be expressed as a similar matrix-vector product. Exploiting this structure, we use a series of Toeplitz matrix products (which are known to have efficient algorithms via FFTs [19]) to compute all these commitments. We give the formal description in Fig. 12.

Computing π_{22} for all points. We observe that, for $i, j \in \{0, \ldots, q-1\}$, if we consider Eq. (3) for the point (ω^i, ω^j), and then put $X_1 = \omega^i$, we get:

$$q_{22}^{(i,j)}(X_2) = \frac{h(\omega^i, X_2) - (X_2 - \omega^j) h_2'(\omega^i, \omega^j) - h(\omega^i, \omega^j)}{(X_2 - \omega^j)^2}$$

Since all terms with degree lower than 2 will be part of the remainder (and not affect the quotient $q_{22}^{(i,j)}$), we can ignore them, and then expand $h(\omega^i, X_2)$ to get:

$$\begin{aligned} q_{22}^{(i,j)}(X_2) &= \frac{\Sigma_{u \in \{0,\ldots,d\}} h_u(X_2) \cdot (\omega^i)^u}{(X_2 - \omega^j)^2} \\ &= \Sigma_{u \in \{0,\ldots,d\}} \frac{h_u(X_2)}{(X_2 - \omega^j)^2} \cdot (\omega^i)^u \\ &= \Sigma_{u \in \{0,\ldots,d\}} \hat{q}_2^{(u,j)}(X_2) \cdot (\omega^i)^u \end{aligned} \tag{5}$$

Here, we use $\hat{q}_2^{(u,j)}$ to denote the quotient of $h_u(X_2)$ when divided by $(X_2 - \omega^j)^2$. Based on the above, we can compute the $\pi_{22}^{(i,j)}$ proofs for all points in the following two steps: (i) for each $u \in \{0, \ldots, d\}$, compute commitments to all quotients of h_u, i.e. $\hat{\pi}_2^{u,j} = [\hat{q}_2^{(u,j)}(\tau_2)]_1$ for all $j \in \{0, \ldots, q-1\}$. Then, (ii) for each $j \in \{0, \ldots, q-1\}$, we can compute the proofs $\pi_{22}^{(i,j)}$ for all $i \in \{0, \ldots, q-1\}$ by running a Discrete Fourier transform over $\{\hat{\pi}_2^{0,j}, \ldots, \hat{\pi}_2^{d,j}\}$. Hence, the problem now boils down to step (i) for which we can rely on techniques similar to those we used for computing π_{11} proofs. Specifically, we can simplify Eq. (4) for univariate polynomials, and then use the same three steps. The formal description of the full algorithm can be found in Fig. 12.

Computing π_{12} for all points. Recall that, for $i \in \{0, \ldots, q-1\}$, Lem. 4.1 implies:

$$q_1^{(i)}(X_1, X_2) = \quad (X_1 - \omega^i) q_{11}^{(i,j)}(X_1, X_2) + (X_2 - \omega^j) q_{12}^{(i,j)}(X_2) + q_1^{(i)}(\omega^i, \omega^j)$$

We can put $X_1 = \omega^i$ and re-arrange terms to get:

$$q_{12}^{(i,j)}(X_2) = \frac{\left(q_1^{(i)}(\omega^i, X_2) - q_1^{(i)}(\omega^i, \omega^j)\right)}{X_2 - \omega^j}$$

In Lem. A.2, we show that,

$$\begin{aligned} q_1^{(i)}(X_1, X_2) = & \; h_d(X_2) \cdot X_1^{d-1} + \\ & (h_{d-1}(X_2) + \omega^i \cdot h_d(X_2)) \cdot X_1^{d-2} + \\ & \quad \vdots \\ & (h_1(X_2) + \omega^i h_2(X_2) + \omega^{2i} h_3(X_2) + \ldots + \omega^{(d-1)i} h_d(X_2)) \cdot X_1^0 \end{aligned} \tag{6}$$

Combining the two equations above leads to,

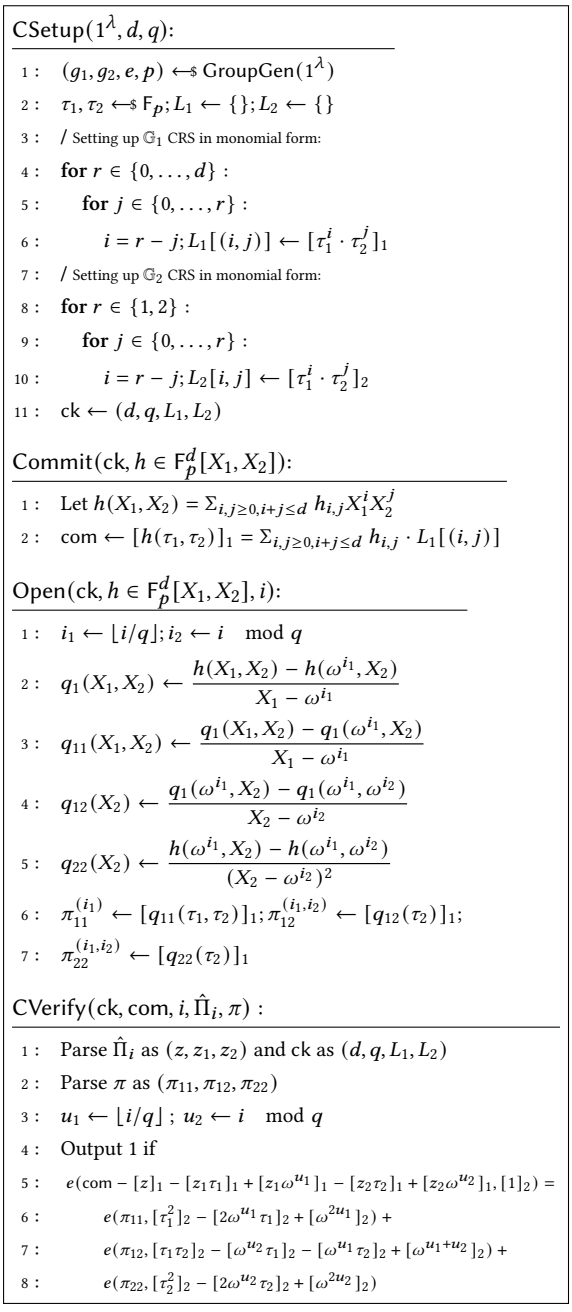
$$\begin{aligned} q_{12}^{(i,j)}(X_2) = & \\ & \frac{\begin{pmatrix} (h_d(X_2) - h_d(\omega^j)) \cdot \omega^{i(d-1)} + \\ (h_{d-1}(X_2) - h_{d-1}(\omega^j) + \omega^i \cdot (h_d(X_2) - h_d(\omega^j))) \cdot \omega^{i(d-2)} + \\ \vdots \\ (\Sigma_{u \in [d]} \omega^{(u-1)i} (h_u(X_2) - h_u(\omega^j))) \end{pmatrix}}{X_2 - \omega^j} \\ & = \frac{\Sigma_{u \in [d]} u \cdot \omega^{i(u-1)} \cdot (h_u(X_2) - h_u(\omega^j))}{X_2 - \omega^j} \\ & = \Sigma_{u \in [d]} u \cdot \omega^{i(u-1)} \cdot \frac{h_u(X_2) - h_u(\omega^j)}{X_2 - \omega^j} \\ & = \Sigma_{u \in [d]} u \cdot \omega^{i(u-1)} \cdot \hat{q}_1^{(u,j)}(X_2) \end{aligned} \tag{7}$$

Here, we use $\hat{q}_1^{u,j}$ to denote the quotient of $h_u(X_2) - h_u(\omega^j)$ when divided by $X_2 - \omega^j$. Based on the above expression, we can compute the proofs $\pi_{12}^{(i,j)}$ for all points in three steps: (i) for each $u \in [d]$, we use the Feist-Khovratovich technique [19] to compute the commitments to all quotients for $h_u(X_2)$, i.e. $\hat{\pi}_1^{(u,j)} = [\hat{q}_1^{(u,j)}(\tau_2)]_1$ for all $j \in \{0, \ldots, q-1\}$, (ii) then, for each $u \in [d]$, we scale the commitments $\hat{\pi}_1^{(u,j)}$ by u, i.e. $\hat{\pi}_1^{(u,j)} \leftarrow u \cdot \hat{\pi}_1^{(u,j)}$. Lastly, (iii) for any $j \in \{0, \ldots, q-1\}$, we can get $\pi_{12}^{(i,j)}$ for all $i \in \{0, \ldots, q-1\}$ with a Discrete Fourier transform over the scaled commitments $\hat{\pi}_1^{(1,j)}, \ldots, \hat{\pi}_1^{(d,j)}$. We make some optimizations, by observing that we can re-use many of the computations done for π_{22} proofs. Fig. 12 presents the full algorithm.

4.2.3 Local correction for openings. Recall that to allow nodes to repair their chunks of the encoding, we need to also support repair for the opening proofs. We now discuss how we can efficiently do local correction for openings for our commitment scheme PSTMult.

Repairing $\pi_{11}^{(i)}$. Recall that, in Sec. 4.1, to repair the evaluation of the bivariate polynomial h at a particular point (ω^i, ω^j), we rely on evaluations at $d+1$ points on a random line L passing through this point. For repairing π_{11}, we take two cases based on the line L. If the line is parallel to the y-axis, i.e. $L(\gamma) = (\omega^i, a\gamma + b)$, then, this is trivial because the opening proof $\pi_{11}^{(i)}$ is the same for all points on this line. We now consider the more interesting case of arbitrary lines wherein the problem boils down to repairing $\pi_{11}^{(i)}$ given opening proofs for $d+1$ other x-values, i.e. $\{\pi_{11}^{i_1}, \ldots, \pi_{11}^{i_{d+1}}\}$.

The key insight we use is that, the proofs $\{\pi_{11}^{(i)}\}_{i \in \{0,\ldots,q-1\}}$ can be considered group elements corresponding to 'evaluations' of a polynomial of degree $d-2$. Specifically, recall that we computed these proofs by running a Discrete Fourier transform over certain elements $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{d-1}$ (as in ln. 12 of Fig. 12) – these elements act as the 'coefficients' of



$\mathsf{CSetup}(1^\lambda, d, q)$:

$(g_1, g_2, e, p) \leftarrow\!\$\ \mathsf{GroupGen}(1^\lambda)$

$\tau_1, \tau_2 \leftarrow\!\$\ \mathsf{F}_p; L_1 \leftarrow \{\}; L_2 \leftarrow \{\}$

/ Setting up $\mathbb{G}_1$ CRS in monomial form:

for $r \in \{0, \ldots, d\}$:

  for $j \in \{0, \ldots, r\}$:

    $i = r - j; L_1[(i,j)] \leftarrow [\tau_1^i \cdot \tau_2^j]_1$

/ Setting up $\mathbb{G}_2$ CRS in monomial form:

for $r \in \{1, 2\}$:

  for $j \in \{0, \ldots, r\}$:

    $i = r - j; L_2[i,j] \leftarrow [\tau_1^i \cdot \tau_2^j]_2$

$\mathsf{ck} \leftarrow (d, q, L_1, L_2)$

$\mathsf{Commit}(\mathsf{ck}, h \in \mathsf{F}_p^d[X_1, X_2])$:

Let $h(X_1, X_2) = \Sigma_{i,j \geq 0, i+j \leq d}\ h_{i,j} X_1^i X_2^j$

$\mathsf{com} \leftarrow [h(\tau_1, \tau_2)]_1 = \Sigma_{i,j \geq 0, i+j \leq d}\ h_{i,j} \cdot L_1[(i,j)]$

$\mathsf{Open}(\mathsf{ck}, h \in \mathsf{F}_p^d[X_1, X_2], i)$:

$i_1 \leftarrow \lfloor i/q \rfloor; i_2 \leftarrow i \mod q$

$q_1(X_1, X_2) \leftarrow \dfrac{h(X_1, X_2) - h(\omega^{i_1}, X_2)}{X_1 - \omega^{i_1}}$

$q_{11}(X_1, X_2) \leftarrow \dfrac{q_1(X_1, X_2) - q_1(\omega^{i_1}, X_2)}{X_1 - \omega^{i_1}}$

$q_{12}(X_2) \leftarrow \dfrac{q_1(\omega^{i_1}, X_2) - q_1(\omega^{i_1}, \omega^{i_2})}{X_2 - \omega^{i_2}}$

$q_{22}(X_2) \leftarrow \dfrac{h(\omega^{i_1}, X_2) - h(\omega^{i_1}, \omega^{i_2})}{(X_2 - \omega^{i_2})^2}$

$\pi_{11}^{(i_1)} \leftarrow [q_{11}(\tau_1, \tau_2)]_1; \pi_{12}^{(i_1, i_2)} \leftarrow [q_{12}(\tau_2)]_1;$

$\pi_{22}^{(i_1, i_2)} \leftarrow [q_{22}(\tau_2)]_1$

$\mathsf{CVerify}(\mathsf{ck}, \mathsf{com}, i, \hat{\Pi}_i, \pi)$:

Parse $\hat{\Pi}_i$ as (z, z_1, z_2) and ck as (d, q, L_1, L_2)

Parse π as $(\pi_{11}, \pi_{12}, \pi_{22})$

$u_1 \leftarrow \lfloor i/q \rfloor$; $u_2 \leftarrow i \mod q$

Output 1 if

$e(\mathsf{com} - [z]_1 - [z_1\tau_1]_1 + [z_1\omega^{u_1}]_1 - [z_2\tau_2]_1 + [z_2\omega^{u_2}]_1, [1]_2) =$

  $e(\pi_{11}, [\tau_1^2]_2 - [2\omega^{u_1}\tau_1]_2 + [\omega^{2u_1}]_2) +$

  $e(\pi_{12}, [\tau_1\tau_2]_2 - [\omega^{u_2}\tau_1]_2 - [\omega^{u_1}\tau_2]_2 + [\omega^{u_1+u_2}]_2) +$

  $e(\pi_{22}, [\tau_2^2]_2 - [2\omega^{u_2}\tau_2]_2 + [\omega^{2u_2}]_2)$

Fig. 11. The setup, commit, opening and verify algorithms for our commitment scheme PSTMult.

$\mathsf{BatchOpen}(\mathsf{ck}, h \in \mathsf{F}_p^d[X_1, X_2])$:

1 : Let $h(X_1, X_2) = \Sigma_{i,j \geq 0, i+j \leq d}\ h_{i,j} X_1^i X_2^j$

2 : Parse ck as (d, q, L_1, L_2)

3 : / Computing $\pi_{11}^{(i)}$ for all $i \in \{0, \ldots, q-1\}$:

4 : Let $h(X_1, X_2) = \Sigma_{u=0}^{d} h_u(X_2) \cdot X_1^u$

5 : $\forall\ u \in \{0, \ldots, d\}$,

6 : Let $h_u(X_2) = \Sigma_{j=0}^{d} h_{u,j} \cdot X_2^j$ / $h_{u,j} = 0$ for all $j > d - u$

7 : $\boldsymbol{w} := [[0]_1 \ldots [0]_1]^T \in \mathbb{G}_1^d$

8 : $\forall\ i \in \{0, \ldots, d-1\}$:

9 : $$\boldsymbol{w}^{(i)} := \begin{bmatrix} h_{d,i} & h_{d-1,i} & \ldots & h_{1,i} \\ 0 & h_{d,i} & \ldots & h_{2,i} \\ . & . & \ldots & . \\ . & . & \ldots & . \\ 0 & 0 & \ldots & h_{d,i} \end{bmatrix} \cdot \begin{bmatrix} L_1[d-1, i] \\ . \\ . \\ L_1[0, i] \end{bmatrix}$$

10 : $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{w}^{(i)}$

11 : $\forall\ i \in \{0, \ldots, d-1\} : \boldsymbol{w}_i \leftarrow i \cdot \boldsymbol{w}_i$

12 : $\left[\pi_{11}^{(0)}, \ldots, \pi_{11}^{(q-1)}\right] := \mathsf{DFT}_{\mathbb{G}_1}(q, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{d-1}, [0]_1, \ldots, [0]_1)$

13 : / Computing $\pi_{12}^{(i,j)}$ and $\pi_{22}^{(i,j)}$ for all $i, j \in \{0, \ldots, q-1\}$:

14 : $\forall u \in \{0, \ldots, d\}$:

15 : $$\boldsymbol{y}^{(u)} := \begin{bmatrix} h_{u,d} & h_{u,d-1} & \ldots & h_{u,1} \\ 0 & h_{u,d} & \ldots & h_{u,2} \\ . & . & \ldots & . \\ . & . & \ldots & . \\ 0 & 0 & \ldots & h_{u,d} \end{bmatrix} \cdot \begin{bmatrix} L_1[0, D-1] \\ . \\ . \\ L_1[0, 0] \end{bmatrix}$$

16 : $\left[\hat{\pi}_1^{(u,0)}, \ldots, \hat{\pi}_1^{(u,q-1)}\right] := \mathsf{DFT}_{\mathbb{G}_1}(q, \boldsymbol{y}_0^{(u)}, \ldots, \boldsymbol{y}_{d-1}^{(u)}, [0]_1, \ldots, [0]_1)$

17 : $\forall j \in \{0, \ldots, q-1\} : \hat{\pi}_1^{(u,j)} \leftarrow u \cdot \hat{\pi}_1^{(u,j)}$

18 : $\forall i \in \{0, \ldots, d-1\} : \boldsymbol{y}_i^{(u)} \leftarrow i \cdot \boldsymbol{y}_i^{(u)}$

19 : $\left[\hat{\pi}_2^{(u,0)}, \ldots, \hat{\pi}_2^{(u,q-1)}\right] := \mathsf{DFT}_{\mathbb{G}_1}(q, \boldsymbol{y}_1^{(u)}, \ldots, \boldsymbol{y}_{d-1}^{(u)}, [0]_1, \ldots, [0]_1)$

20 : $\forall j \in \{0, \ldots, q-1\}$:

21 : $\left[\pi_{12}^{0,j}, \ldots, \pi_{12}^{q-1,j}\right] := \mathsf{DFT}_{\mathbb{G}_1}(q, \hat{\pi}_1^{1,j}, \ldots, \hat{\pi}_1^{d,j}, [0]_1, \ldots, [0]_1)$

22 : $\left[\pi_{22}^{0,j}, \ldots, \pi_{22}^{q-1,j}\right] := \mathsf{DFT}_{\mathbb{G}_1}(q, \hat{\pi}_2^{0,j}, \ldots, \hat{\pi}_2^{d,j}, [0]_1, \ldots, [0]_1)$

Fig. 12. The batch opening algorithm for our commitment scheme PSTMult. We use DFT as a subroutine – this algorithm computes the discrete Fourier transform over roots of unity of order q. As shown in [19], the Toeplitz matrix vector products (in lns. 9 and 15) can be implemented in time $O(d \log(d))$ via Discrete fourier transforms.

the polynomial and the proofs are 'evaluations' at all roots of unity. Hence, if we have any $d-1$ 'evaluations' i.e. proofs, we can do lagrange interpolation in the exponent to compute the proof at any point ω^i. We give the formal algorithm in Fig. 13.

Repairing $\pi_{12}^{(i,j)}$. To repair $\pi_{12}^{(i,j)}$ given $d+1$ points on a line, we show in Lem. A.4 that these proofs can be seen as 'evaluations' of a bivariate polynomial with 'coefficients' derived from the $\boldsymbol{y}^{(u)}$ vectors (defined in ln. 15 of Fig. 12). Intuitively, this follows from the fact that we compute these proofs by running a series of Discrete fourier transforms analogous to how one computes evaluations of a bivariate polynomial – as described in Sec. 4.1.

We show that the total degree of this polynomial is bounded by $d-2$, which means that when we restrict this polynomial to any line, the resulting univariate polynomial would also have degree $d-2$. Hence, given 'evaluations' at any $d-1$ points i.e. opening proofs, on a line passing through (ω^i, ω^j), we can use lagrange interpolation in the exponent to get the 'evaluation' i.e. proof at (ω^i, ω^j). Fig. 13 gives the formal algorithm.

Repairing $\pi_{22}^{(i,j)}$. For repairing the last opening proof, we make a similar observation in Lem. A.5, that these proofs can be considered group elements corresponding to 'evaluations' of a bivariate polynomial with 'coefficients' derived from the $\boldsymbol{y}^{(u)}$ vectors. This again comes from the fact that we compute these proofs by running a series of Discrete Fourier transforms analogous to evaluating a bivariate polynomial in the exponent.

We show that the total degree of this bivariate polynomial is also bounded by $d-2$, so, given the 'evaluations' i.e. proofs of any $d-1$ points on a line passing through (ω^i, ω^j), we can use lagrange interpolation in the exponent to get the 'evaluation' i.e. proof at (ω^i, ω^j). Fig. 13 gives the formal algorithm.

We now present the correctness and security analysis. The proofs for all the theorems can be found in App. A.

Thm. 4.2 below proves that our construction has zero correctness error.

Theorem 4.2. *The commitment scheme* PSTMult *is correct with* $\eta = 0$.

Thm. 4.3 below proves that the all the opening proofs computed by our batch open algorithm are indeed correct.

Theorem 4.3. *The commitment scheme* PSTMult *is* η_b*-correct with respect to batch openings, with* $\eta_b = 0$.

Thm. 4.4 below proves correctness of local correction of openings.

Theorem 4.4. *For any* PPT *adversary* $\mathcal{A}$*, there exist* PPT *adversaries* $\mathcal{B}_1, \mathcal{B}_2$ *such that:*

$$\mathsf{Adv}^{\mathrm{loc-cor-open}}_{\mathcal{A},\mathsf{PSTMult}}(\lambda) \leq \mathsf{Adv}^{\mathrm{extr}}_{\mathcal{B}_1,\mathsf{PSTMult}}(\lambda) + \mathsf{Adv}^{\mathrm{pos-bind}}_{\mathcal{B}_2,\mathsf{PSTMult}}(\lambda)$$

We now move on to proving security for our scheme. Specifically, we prove position-binding and extractability. The other binding properties such as code-binding and reconstruction-binding are implied by these two properties since our commitment algorithm is also deterministic, as proven in [31, Lemmas 30,2].

Position binding. Thm. 4.6 below proves position binding of our commitment scheme, by relying on the asymmetric ℓ-strong Bilinear Diffie-Hellman (also called asymmetric ℓ-SBDH) assumption that we formalize in Def. 4.5.

Definition 4.5. We say that the asymmetric ℓ-SBDH problem is hard if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible in λ:

$$\mathsf{Adv}^{\ell-\mathrm{sbdh}}_{\mathcal{A}}(\lambda) := \Pr\left[\begin{array}{c} ([1]_1, [1]_2, e, p) \leftarrow\!\$\ \mathsf{GroupGen}(1^\lambda) \\ \tau \leftarrow\!\$\ \mathsf{F}_p \\ (c, \frac{1}{\tau+c} \cdot e([1]_1, [1]_2)) \leftarrow\!\$\ \mathcal{A}\left(\begin{array}{c} e, p, \\ \{[\tau^0]_1, [\tau]_1, \ldots, [\tau^\ell]_1\} \\ \{[\tau^0]_2, [\tau]_2, \ldots, [\tau^\ell]_2\} \end{array}\right) \end{array}\right]$$

$\mathsf{CLocalCorrect}(\mathsf{ck}, \mathsf{com}, i, \hat{\Pi}, \{\pi_j\}_{j\in\hat{\Pi}})$

Parse $\hat{\Pi}$ as $\{\hat{\Pi}_{j_1}, \ldots, \hat{\Pi}_{j_{d+1}}\}$

$\forall u \in [d+1], j_{u,1} \leftarrow \lfloor j_u/q \rfloor$; $j_{u,2} \leftarrow j_u \mod q$

$\forall u \in [d+1]$, Parse π_{j_u} as $(\pi_{11}^{(j_{u,1})}, \pi_{12}^{(j_{u_1}, j_{u_2})}, \pi_{22}^{(j_{u_1}, j_{u_2})})$

$i_1 \leftarrow \lfloor i/q \rfloor$; $i_2 \leftarrow i \mod q$

/ Repairing $\pi_{11}^{(i_1)}$:

if $\exists\, u \in [d+1]$ s.t. $j_{u,1} = i_1$

  then $\pi_{11}^{(i_1)} \leftarrow \pi_{11}^{(j_{u,1})}$

else $\pi_{11}^{(i_1)} \leftarrow \Sigma_{u\in[d-1]} \lambda^{i_1}_{j_{u,1},\{j_{1,1},\ldots,j_{d-1,1}\}} \cdot \pi_{11}^{j_{u,1}}$ where

$$\lambda^{i_1}_{j_{u,1},\{j_{1,1},\ldots,j_{d-1,1}\}} = \prod_{v\in[d-1]\setminus\{u\}} \frac{\omega^{j_{v,1}} - \omega^{i_1}}{\omega^{j_{v,1}} - \omega^{j_{u,1}}}$$

/ Repairing $\pi_{12}^{(i_1,i_2)}$ and $\pi_{12}^{(i_1,i_2)}$:

Let $L(\gamma) := (a\gamma + \omega^{i_1}, a'\gamma + \omega^{i_2})$ / Line through all points

$\forall u \in [d+1]$: Define γ_u s.t. $(\omega^{j_{u,1}}, \omega^{j_{u,2}}) = L(\gamma_u)$

$$\lambda^{0}_{\gamma_u,\{\gamma_1,\ldots,\gamma_{d-1}\}} := \prod_{v\in[d-1]\setminus\{u\}} \frac{\gamma_v}{\gamma_v - \gamma_u}$$

$\pi_{12}^{(i_1,i_2)} \leftarrow \Sigma_{u\in[d-1]} \lambda^{0}_{\gamma_u,\{\gamma_1,\ldots,\gamma_{d-1}\}} \cdot \pi_{12}^{(j_{u,1}, j_{u_2})}$

$\pi_{22}^{(i_1,i_2)} \leftarrow \Sigma_{u\in[d-1]} \lambda^{0}_{\gamma_u,\{\gamma_1,\ldots,\gamma_{d-1}\}} \cdot \pi_{22}^{(j_{u,1}, j_{u_2})}$

Fig. 13. The algorithm for Local correction of opening proofs for our commitment scheme PSTMult.

Theorem 4.6. *For every* PPT *adversary* $\mathcal{A}$*, there exists* PPT *adversaries* $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ *such that*

$$\mathsf{Adv}^{\mathrm{pos-bind}}_{\mathcal{A},\mathsf{PSTMult}}(\lambda) \le q^2 \cdot \begin{pmatrix} \mathsf{Adv}^{\mathrm{d-sbdh}}_{\mathcal{B}_1}(\lambda) + \\ \mathsf{Adv}^{\mathrm{d-sbdh}}_{\mathcal{B}_2}(\lambda) + \\ + \mathsf{Adv}^{\mathrm{d-sbdh}}_{\mathcal{B}_3}(\lambda) + 1/p \end{pmatrix}$$

where d *is the upper bound on the degree of polynomials committed to in* PSTMult.

Extractability. We prove that our commitment scheme is extractable in the Algebraic Group Model (AGM), similar to [31, App.E.3]. Informally, in the AGM, for every group element that the adversary outputs, it also has to provide the coefficients describing this group element as a linear combination of the group elements in the commitment key. These coefficients are exactly the coefficients of the committed polynomial. We formally state this in Thm. 4.7, whose proof can be found in App. A.

Theorem 4.7. *In the Algebraic Group Model (AGM), for all adversaries* $\mathcal{A}$*, we have:*

$$\mathsf{Adv}^{\mathrm{extr}}_{\mathcal{A},\mathsf{PSTMult}}(\lambda) = 0$$

4.3 Reducing the network load for Dispersal

Recall that for our comparisons, we assume that a peer-to-peer gossip protocol is used to disperse the encoded data. Since there is one gossip subnet for each chunk of the data, the number of gossip subnets for our construction would be

q^m, i.e. the length of the encoding (here q is the number of evaluation points and m is the number of derivatives, as described in Sec. 4.1). Based on the parameters described in Tab. 3, the number of subnets turns out to be $96^2 = 9216$, which cannot be supported by the network (see [23] for details).

To reduce the number of subnets, we use a simple batching technique – we split the data into Y rows, and each row is individually encoded with the multiplicity code. A single chunk of the encoding now contains one symbol from all the encoded rows. For example, the first symbol of all the rows becomes the first chunk, and so on. This allows us to reduce the number of subnets since the number of chunks only depends on the length of the encoding of a single row. Note that each chunk is now bigger by a factor of Y, meaning that node storage and bandwidth for repair also increases by the same factor.

Lastly, commitment to the data now contains a separate commitment to each row using the scheme described in Sec. 4.2. The opening proof for a chunk of the encoding then contains a separate opening proof for each row. We choose $Y = 8$, i.e. split data into 8 rows, so that we still get efficient repair while reducing the number of subnets by a factor of eight. A detailed comparison of this technique with the Ethereum's Fulu DAS can be found in Tabs. 1 and 2.

5 Evaluation

We implemented our DAS scheme and present experimental results in this section.

5.1 Implementation

We implemented our encoding and commitment scheme in C. Note that our implementation and results focus on the construction without batching, but it can be easily modified to incorporate the batching technique described in Sec. 4.3.

We provide the same interface as c-kzg-4844 [25], which is an implementation of Ethereum's proposed scheme. Specifically, we implement the encoding algorithm given in Fig. 9. We implement the Interpolate algorithm from [60, Sec. 10.2], and for FFTs, we rely on the implementation in [25].

For the commitment, we implement the CSetup, Commit, CVerify and BatchOpen algorithms. We use BLS12-381, a pairing-friendly elliptic curve, implemented in the blst library [48]. To be able to use FFTs for encoding and commitments, we round q to the next power of two, i.e. 128. Note that, for the Toeplitz matrix-vector products in the BatchOpen algorithm (lns. 9 and 15), we implement two algorithms—(i) a naive algorithm which uses a multi-scalar multiplication (MSM) to compute each element of the matrix-vector product, (ii) a Discrete Fourier transform-based algorithm as given in [19]. While the DFT-based approach runs in time $O(q \log(q))$, We observe that for small matrices (of size 128 or less), the first algorithm is more efficient, so we report numbers based on the naive MSM-based algorithm.

5.2 Results

Tab. 4 compares our scheme with Ethereum's Fulu DAS proposal (since that is the only DAS scheme with an implementation). All the experiments were conducted on a single core on an Apple M1 Pro machine. We measure the runtimes based on CLOCK_MONOTONIC, which reports wall clock times, and we report averages over five runs.

Encoding Comparison. We observe that it takes about 47ms to encode with our multiplicity code, as compared to 3ms for encoding with a Reed-Solomon code as in Ethereum. This is because for encoding a bivariate polynomial, we need to use the Interpolate algorithm which requires $O(d \log^2(d))$ field operations. While our encoding is relatively slower, we note that it is still quite reasonable.

Metric	Ethereum Fulu DAS	MultDAS
Encoding time	2.6ms $O(B\log(B))$	46.1ms $O(B\log^2(B))$
CRS size	192 KB (4096 $\mathbb{G}_1$ + 1 $\mathbb{G}_2$)	196.7 KB (4186 $\mathbb{G}_1$ + 5 $\mathbb{G}_2$)
Commitment size	48 bytes (1 $\mathbb{G}_1$)	48 bytes (1 $\mathbb{G}_1$)
Proof size	48 bytes (1 $\mathbb{G}_1$)	144 bytes (3 $\mathbb{G}_1$)
Commitment time	41ms (MSM of size 4096)	44ms (MSM of size 4186)
Verification time	1.15ms (2 pairings)	2.9ms (4 pairings)
BatchOpen time	0.3s $O(B\log(B))$	20.3s $O(B\log(B))$
Multi-threaded BatchOpen time	0.3s	0.38s

Table 4. Comparing our construction MultDAS (without batching) with Ethereum, for data of size 4096 field elements. Here, B denotes the size of the data, and MSM stands for Multi-scalar multiplication. Note that while the BatchOpen algorithm for our scheme is slow when run on a single core, it is highly parallelizable. We estimate that it can run in about 0.38s on a machine with 96 cores.

Commitment Comparison. *CRS.* For our commitment scheme, the CRS includes terms of the form $[\tau_1^i \tau_2^j]_1$ for all i, j such that $i + j \leq d$. This leads to 4186 $\mathbb{G}_1$ elements in the CRS, and five $\mathbb{G}_2$ elements which are used for verification. Overall, the CRS size is only 2% larger than that of Ethereum.

Commitment Size and Proof Verification. A commitment in our scheme PSTMult is one $\mathbb{G}_1$ element, which is the same as Ethereum. The proof for our scheme includes three $\mathbb{G}_1$ elements, which is three times as large as a proof in the Ethereum scheme. Verifying a proof requires four pairings, whereas Ethereum only requires two. We also validate this empirically – our verification takes about 2.5× more time than Ethereum.

Commitment Time. To commit in our scheme, we simply need to do a multi-scalar multiplication, wherein we multiply $L_1[(i, j)]$ by the corresponding coefficient $h_{i,j}$ of the polynomial. Since our degree is 90, the length of the MSM is the number of coefficients, which is $\binom{92}{2} = 4186$. The commit algorithm for Ethereum also does a multi-scalar multiplication of size 4096. Empirically, we observe that this leads to only 7% increase in the runtime for the Commit algorithm.

Batch Opening. Computing the opening proofs for all points takes about 60× more sequential runtime than Ethereum, even though the complexity for both the schemes is $O(R\log(R))$. This is because we need to do a lot of small FFT and multi-scalar multiplications of sizes d and q (both of which are $\Omega(\sqrt{R})$). While this is slower, we do note two things:

- In systems such as Ethereum [26], each row of the data (also called a blob) is "owned" by a different entity (rollup operator). Hence, the work of encoding, commitment and computing all the opening proofs for each row can be done by the party who owns it. This natural parallelization greatly reduces the workload of the dispersor, i.e., block

producer. Furthermore, this computation is not on the critical path of consensus, but can take place as part of the "mempool" of blobs, and thus does not entail an increase in Ethereum's block inter-production time of 12s.

- Moreover, our BatchOpen algorithm is highly parallelizable. Specifically, the d Toeplitz matrix-vector products (in lns. 8 to 9) can be executed in parallel. Similarly, the $d + 1$ iterations of the for loop in lns. 14 to 19 can also be executed in parallel. Lastly, the $2q$ Discrete Fourier transforms in lns. 20 to 22 can all be done in parallel. With these optimizations, we estimate the multi-threaded runtime of BatchOpen on a 96-core machine to be around 0.38s, which is reasonable. Note that we assume that the FFT and the multi-scalar multiplications themselves are run sequentially, but we can run multiple FFTs in parallel.

6 Extensions

While our DAS construction achieves efficient repair, there are many interesting extensions worth exploring:

Reducing number of subnets for dispersal. In Sec. 4.3, we proposed a simple batching technique which allows us to reduce the number of subnets required for dispersal in our scheme. It would be interesting to see if there are more efficient ways of achieving this. For instance, could we commit to the data by simply taking a random linear combination of the commitments to individual rows? This would reduce the size of the commitment and opening proofs for each chunk.

Supporting other types of repair for multiplicity codes. As mentioned in Sec. 4.1, multiplicity codes support another method of local correction. Specifically, for a bivariate polynomial, to repair the evaluations for a point, we can sample any two lines passing through this point, and then evaluations of the polynomial h and its derivatives at any $\lceil d/2 \rceil$ points on both lines is enough to repair this point (see [38] for a full description). Supporting this local correction algorithm for our erasure code commitment (described in Sec. 4.2) is another interesting extension.

Generalizing commitment to multivariates. Note that while we described our commitment scheme for the case of bivariate polynomials, we believe it can be readily extended to commit to multivariate polynomials as well. We leave this task as a potential future extension.

7 Additional Related Work

Erasure-Correcting Codes and Consensus. Erasure-correcting codes have been suggested for a variety of purposes in the context of (state-machine replication) consensus. All of these approximately rest on a primitive such as VDS sketched in Sec. 1 and Fig. 2 [55]. One use of erasure-coding in consensus is to reduce the confirmation latency of consensus, by taking the dissemination of the full payload off the critical path to the consensus decision. Instead, nodes receive only erasure-coded chunks of payload during consensus and before they vote for proposed (references to) payloads, and the full payloads are reconstructed in a background process. This approach was first advocated for crash-fault tolerant protocols like Paxos [47] and subsequently Raft [64], before being adapted to Byzantine-fault tolerant protocols [64] and leaderless protocols [43, 69].

Another use of erasure-coding is to horizontally scale consensus protocols, as advocated in Sec. 1. This approach was particularly studied in the context of blockchains [1, 12, 18, 49, 52] and can be found in systems such as Ethereum [52], NEAR [3], and Celestia [40].

Finally, erasure-coding was proposed in the theory of distributed computing to improve the communication complexity of consensus protocols [4, 5, 13, 46, 49], and to alleviate the throughput bottleneck posed by the limited communication bandwidth of nodes that act as leaders, such as in Solana's Turbine [6] and Alpenglow's Rotor [7].

Vote-Based Data Availability (VDA). VDA as a special form of VDS originates with the asynchronous verifiable information dispersal (AVID) protocol [10]. Subsequent work [33] has improved the complexity of VID or adjusted the modality by which availability is verified [56]. Some of the efficiency improvements result from weakening VID's security guarantees [49, 66] in ways that are still compatible with applications of interest. VDA is found in blockchain systems such as rollups' data availability committees [49], EigenDA [41, 42], or Espresso [9, App. A]. Recent work [22] has extended VDA to the permissionless setting, and studied the incentives of rollups' data availability committees [58].

Data Availability Sampling (DAS). Early works [35] advocate the use of sampling to interactively and succinctly verify the availability of a (large) file deposited at a storage node. This theme was later reprised in the blockchain context, where sampling was proposed for light clients to verify the availability (if not the validity) of blocks [2]. That work used the "blackboard model" [50], which has been criticized as unrealistic [36, 50]. In contrast, a security notion of DAS was made precise in a stronger model in [31] using an extractor-based definition, which we also build on with Def. 2.2. Subsequent works have explored different commitment schemes such as FRI [30], and different erasure-correcting codes such as LDPC codes [68] or interleaved Reed–Solomon codes [29]. Unlike these works, the present paper is the first to study LCCs and polynomial commitment schemes matched to LCCs, to improve the repair properties of the resulting scheme. Recent work [17] has studied synergies between polynomial commitment schemes used in data availability and in rollups' validity proofs.

State-of-the-art implementation proposals for various stages of DAS in Ethereum as part of the Danksharding vision include PeerDAS [54], SubnetDAS [26], and FullDAS [24].

Polynomial Commitment Schemes. SoCC [51] introduced the PST polynomial commitment scheme, and also described how to open the commitment to evaluations of derivatives. But, for opening even a single derivative, the proof size grows with the degree of the polynomial – this is in contrast to our scheme, where the proof size is constant, i.e., just three group elements (independent of the degree), for opening both first partial derivatives of a bivariate polynomial. Moreover, they do not support batch opening or local correction of opening proofs. [70] focuses on commitments for univariate polynomials, with efficient batch opening (called one-to-many prover in their work). [71] studies commitments to multivariate polynomials with efficient batch opening, but it is unclear how to support (i) opening the derivative evaluations, and (ii) local correction of opening proofs.

8 Conclusion

We formalized the security properties needed of DAS schemes for horizontally scaling consensus. A particular focus has been on the notion of efficient repair, which motivated many of the choices of Ethereum's Danksharding vision. We presented a new DAS scheme designed with efficient repair in mind, based on locally-correctable multiplicity codes and a new multivariate polynomial commitment scheme that supports fast batch opening of codeword symbols and efficient repair of opening proofs. The proposed scheme improves upon the state-of-the-art Fulu DAS design in storage overhead and repair bandwidth/locality, with acceptable compromises in dispersal cost.

There are a lot of interesting avenues for future work, including: (i) If and how other locally-correctable codes (LCCs) can be employed for efficient repair, such as lifted Reed–Muller codes [34, 44], or expander codes [32]. This would also require coming up with corresponding erasure code commitment schemes. (ii) Formalizing the notion of efficient uncoded repair using code symbols repeated across multiple storage nodes, perhaps through the lens of fractional repetition codes [53], and combining it with LCCs. (iii) Devising a mechanism to verify (and possibly incentivize) that

storage nodes store and (on-demand) release the information they are supposed to store, possibly using techniques like proof of retrievability [35], proofs of replication [20, 21], or fair data exchange [59].

Acknowledgments

We thank Noah Citron, Dankrad Feist, and Itzhak (Zachi) Tamo for fruitful discussions.

References

[1] Mustafa Al-Bassam. Lazyledger: A distributed data availability ledger with client-side smart contracts. *CoRR*, abs/1905.09274, 2019. URL: http://arxiv.org/abs/1905.09274, arXiv:1905.09274.

[2] Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin, and Ismail Khoffi. Fraud and data availability proofs: Detecting invalid blocks in light clients. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021*. doi:10.1007/978-3-662-64331-0_15.

[3] Bowen Wang Alex Skidanov, Illia Polosukhin. Nightshade: Near protocol sharding design 2.0, 2024. URL: https://discovery-domain.org/papers/nightshade.pdf.

[4] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. Cryptology ePrint Archive, Paper 2022/776, 2022. URL: https://eprint.iacr.org/2022/776.

[5] Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. Succinct erasure coding proof systems. Cryptology ePrint Archive, Paper 2021/1500, 2021. URL: https://eprint.iacr.org/2021/1500.

[6] Anza. Turbine block propagation. Solana Documentation. URL: https://docs.anza.xyz/consensus/turbine-block-propagation.

[7] Anza. Alpenglow: A new consensus for solana, 2025. Anza Blog. URL: https://www.anza.xyz/blog/alpenglow-a-new-consensus-for-solana.

[8] Daniel Augot, Françoise Levy-dit Vehel, and Cuong M. Ngô. Information sets of multiplicity codes. In *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 2401–2405, 2015. doi:10.1109/ISIT.2015.7282886.

[9] Jeb Bearer, Benedikt Bünz, Philippe Camacho, Binyi Chen, Ellie Davidson, Ben Fisch, Brendon Fish, Gus Gutoski, Fernando Krell, Chengyu Lin, Dahlia Malkhi, Kartik Nayak, Keyao Shen, Alex Xiong, Nathan Yospe, and Sishan Long. The espresso sequencing network: HotShot consensus, tiramisu data-availability, and builder-exchange. Cryptology ePrint Archive, Paper 2024/1189, 2024. URL: https://eprint.iacr.org/2024/1189.

[10] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *Distributed Computing*, 2005.

[11] Kostas Kryptos Chalkias, Charanjit Jutla, Jonas Lindstrom, Varun Madathil, and Arnab Roy. Improved polynomial division in cryptography. Cryptology ePrint Archive, Paper 2024/1279, 2024. URL: https://eprint.iacr.org/2024/1279.

[12] Shir Cohen, Guy Goren, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Proof of availability and retrieval in a modular blockchain architecture. In *Financial Cryptography and Data Security: FC 2023*. doi:10.1007/978-3-031-47751-5_3.

[13] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. 2021. doi:10.1145/3460120.3484808.

[14] Arbitrum Docs. How arbitrum data availability works, 2025. URL: https://docs.arbitrum.io/how-arbitrum-works/data-availability.

[15] EigenDA Docs. EigenDA overview, 2024. URL: https://docs.eigenda.xyz/overview.

[16] StarkEx Docs. Data availability, 2023. URL: https://docs.starkware.co/starkex/con_data_availability.html.

[17] Alex Evans, Nicolas Mohnblatt, and Guillermo Angeris. ZODA: Zero-overhead data availability. Cryptology ePrint Archive, Paper 2025/034, 2025. URL: https://eprint.iacr.org/2025/034.

[18] Dankrad Feist. New sharding design with tight beacon and shard block integration, 2022. URL: https://notes.ethereum.org/@dankrad/new_sharding.

[19] Dankrad Feist and Dmitry Khovratovich. Fast amortized KZG proofs. Cryptology ePrint Archive, Paper 2023/033, 2023. URL: https://eprint.iacr.org/2023/033.

[20] Ben Fisch. Tight proofs of space and replication. In *Advances in Cryptology – EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. doi:10.1007/978-3-030-17659-4_17.

[21] Ben Fisch. Poreps: Proofs of space on useful data. Cryptology ePrint Archive, Paper 2018/678, 2018. URL: https://eprint.iacr.org/2018/678.

[22] Ben Fisch, Arthur Lazzaretti, Zeyu Liu, and Lei Yang. Permissionless verifiable information dispersal (data availability for bitcoin rollups). Cryptology ePrint Archive, Paper 2024/1299, 2024. URL: https://eprint.iacr.org/2024/1299.

[23] Ethereum Foundation. Ethereum Fulu fork specifications. URL: https://github.com/ethereum/consensus-specs/tree/dev/specs/fulu.

[24] Ethereum Foundation. FullDAS - full data availability sampling. URL: https://ethresear.ch/t/fulldas-towards-massive-scalability-with-32mb-blocks-and-beyond/19529.

[25] Ethereum Foundation. C-kzg-4844, 2025. URL: https://github.com/ethereum/c-kzg-4844.

[26] Francesco. SubnetDAS - an intermediate das approach, 2023. URL: https://ethresear.ch/t/subnetdas-an-intermediate-das-approach/17169.

[27] Francesco. Decoupling global liveness and individual safety in DAS, 2024. URL: https://notes.ethereum.org/@fradamt/DAS-security-notions.

[28] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62. Springer, 2018. doi:10.1007/978-3-319-96881-0_2.

[29] Yanpei Guo, Alex Luoyuan Xiong, Wenjie Qu, and Jiaheng Zhang. Data availability for thousands of nodes. Cryptology ePrint Archive, Paper 2025/865, 2025. URL: https://eprint.iacr.org/2025/865.

[30] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. FRIDA: Data availability sampling from fri. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*.

[31] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Foundations of data availability sampling. *IACR Communications in Cryptology*, 1(4), 2025. doi:10.62056/a09qudhdj.

[32] Brett Hemenway, Rafail Ostrovsky, and Mary Wootters. Local correctability of expander codes, 2015. URL: https://arxiv.org/abs/1304.8129, arXiv:1304.8129.

[33] James Hendricks, Gregory R. Ganger, and Michael K. Reiter. Verifying distributed erasure-coded data. In *the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07. doi:10.1145/1281100.1281122.

[34] Lukas Holzbaur, Rina Polyanskaya, Nikita Polyanskii, Ilya Vorobyev, and Eitan Yaakobi. Lifted Reed-Solomon codes and lifted multiplicity codes, 2021. URL: https://arxiv.org/abs/2110.02008, arXiv:2110.02008.

[35] Ari Juels and Burton S. Kaliski Jr. PORs: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07. doi:10.1145/1315245.1315317.

[36] Sreeram Kannan. DA Sampling security guarantees, 2022. URL: https://twitter.com/sreeramkannan/status/1563615609925304320.

[37] Tohru Kohrita and Patrick Towa. Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. Cryptology ePrint Archive, Paper 2023/917, 2023. URL: https://eprint.iacr.org/2023/917, doi:10.1007/s00145-024-09519-0.

[38] Swastik Kopparty, Shubhangi Saraf, and Sergey Yekhanin. High-rate codes with sublinear-time decoding. *J. ACM*, 61(5), September 2014. doi:10.1145/2629416.

[39] Aptos Labs. Quorum Store: How consensus horizontally scales on the Aptos blockchain, 2023. URL: https://medium.com/aptoslabs/quorum-store-how-consensus-horizontally-scales-on-the-aptos-blockchain-988866f6d5b0.

[40] Celestia Labs. Celestia. URL: https://celestia.org/.

[41] Layr Labs. EigenDA protocol architecture: Encoding. URL: https://github.com/Layr-Labs/eigenda/blob/master/docs/spec/src/protocol/architecture/encoding.md.

[42] Layr Labs. EigenDA spec introduction. EigenDA Documentation. URL: https://layr-labs.github.io/eigenda/introduction.html.

[43] Andrew Lewis-Pye and Ehud Shapiro. Morpheus consensus: Excelling on trails and autobahns, 2025. URL: https://arxiv.org/abs/2502.08465, arXiv:2502.08465.

[44] Ray Li and Mary Wootters. Lifted multiplicity codes and the disjoint repair group property, 2020. URL: https://arxiv.org/abs/1905.02270, arXiv:1905.02270.

[45] libp2p. Gossipsub v1.1, 2021. URL: https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/gossipsub-v1.1.md.

[46] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-MVBA: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20. doi:10.1145/3382734.3405707.

[47] Shuai Mu, Kang Chen, Yongwei Wu, and Weimin Zheng. When Paxos meets erasure code: reduce network and storage cost in state machine replication. 2014. doi:10.1145/2600212.2600218.

[48] Supra National. blst, 2025. URL: https://github.com/supranational/blst/tree/master.

[49] Kamilla Nazirkhanova, Joachim Neu, and David Tse. Information dispersal with provable retrievability for rollups. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, AFT '22, pages 180–197, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3558535.3559778.

[50] Joachim Neu. Data availability sampling: From basics to open problems, 2022. URL: https://www.paradigm.xyz/2022/08/das.

[51] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In Amit Sahai, editor, *Theory of Cryptography*, 2013.

[52] Ethereum Roadmap. Danksharding, 2025. URL: https://ethereum.org/en/roadmap/danksharding/.

[53] Salim El Rouayheb and Kannan Ramchandran. Fractional repetition codes for repair in distributed storage systems, 2010. URL: https://arxiv.org/abs/1010.2551, arXiv:1010.2551.

[54] Danny Ryan. PeerDAS – a simpler das approach using battle-tested p2p components, 2023. URL: https://ethresear.ch/t/peerdas-a-simpler-das-approach-using-battle-tested-p2p-components/16541.

[55] Zhirong Shen, Yuhui Cai, Keyun Cheng, Patrick P. C. Lee, Xiaolu Li, Yuchong Hu, and Jiwu Shu. A survey of the past, present, and future of erasure coding for storage systems, 2025. doi:10.1145/3708994.

[56] Peiyao Sheng, Bowen Xue, Sreeram Kannan, and Pramod Viswanath. ACeD: Scalable data availability oracle, 2021. URL: https://arxiv.org/abs/2011.00102, arXiv:2011.00102.

[57] Alex Stokes. blob/acc in 2025. URL: https://hackmd.io/@ralexstokes/blob-acc-2025.

[58] Ertem Nusret Tas and Dan Boneh. Cryptoeconomic security for data availability committees, 2023. URL: https://arxiv.org/abs/2208.02999, arXiv:2208.02999.

[59] Ertem Nusret Tas, István András Seres, Yinuo Zhang, Márk Melczer, Mahimna Kelkar, Joseph Bonneau, and Valeria Nikolaenko. Atomic and fair data exchange via blockchain. Cryptology ePrint Archive, Paper 2024/418, 2024. URL: https://eprint.iacr.org/2024/418, doi:10.1145/3658644.3690248.

[60] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3 edition, 2013.

[61] Dimitris Vyzovitis, Yusef Napora, Dirk McCormick, David Dias, and Yiannis Psaras. GossipSub: Attack-resilient message propagation in the filecoin and eth2.0 networks, 2020. URL: https://arxiv.org/abs/2007.02754, arXiv:2007.02754.

[62] Benedikt Wagner and Francesco D'Amato. Revisiting secure DAS in one and two dimensions. URL: https://ethresear.ch/t/revisiting-secure-das-in-one-and-two-dimensions/22762#p-55355-setting-4-the-realistic-one-18.

[63] Benedikt Wagner and Francesco D'Amato. Revisiting secure DAS in one and two dimensions, 2025. URL: https://ethresear.ch/t/revisiting-secure-das-in-one-and-two-dimensions/22762.

[64] Zizhong Wang, Tongliang Li, Haixia Wang, Airan Shao, Yunren Bai, Shangming Cai, Zihan Xu, and Dongsheng Wang. CRaft: An erasure-coding-supported version of raft for reducing storage cost and network cost. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*. URL: https://www.usenix.org/conference/fast20/presentation/wang-zizhong.

[65] Fei Wu, Thomas Thiery, Stefanos Leonardos, and Carmine Ventre. From competition to centralization: The oligopoly in Ethereum block building auctions, 2024. URL: https://arxiv.org/abs/2412.18074, arXiv:2412.18074.

[66] 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)*. URL: https://www.usenix.org/conference/nsdi22/presentation/yang.

[67] Sergey Yekhanin. *Locally Decodable Codes*, volume 6 of *Foundations and Trends in Theoretical Computer Science*. Now Publishers, 2012. doi:10.1561/0400000030.

[68] Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. Coded Merkle Tree: Solving data availability attacks in blockchains, 2019. URL: https://arxiv.org/abs/1910.01247, arXiv:1910.01247.

[69] Jonathan Zarnstorff, Lucas Lebow, Christopher Siems, Dillon Remuck, Colin Ruiz, and Lewis Tseng. Racos: Improving erasure coding state machine replication using leaderless consensus. 2024. doi:10.1145/3698038.3698511.

[70] Jiaheng Zhang, Tiancheng Xie, Thang Hoang, Elaine Shi, and Yupeng Zhang. Polynomial commitment with a one-to-many prover and applications. In *USENIX Security Symposium*, 2022. URL: https://api.semanticscholar.org/CorpusID:249119947.

[71] Zongyang Zhang, Weihan Li, Yanpei Guo, Kexin Shi, Sherman S. M. Chow, Ximeng Liu, and Jin Dong. Fast RS-IOP multivariate polynomial commitments and verifiable secret sharing. In *the 33rd USENIX Conference on Security Symposium*, SEC '24, USA, 2024.

A Deferred Proofs for PSTMult

Proof of Thm. 4.2. We recall Eq. (2), which was obtained by applying Lem. 4.1 to the committed polynomial h and to its quotients q_1 and q_2. We then use Thm. A.1 to get that, for all $i_1, i_2 \in \{0, \ldots, q-1\}$:

$$\begin{aligned} & h(X_1, X_2) = \\ & (X_1 - \omega^{i_1})^2 q_{11}^{(i_1)}(X_1, X_2) + (X_1 - \omega^{i_1})(X_2 - \omega^{i_2}) q_{12}^{(i_1,i_2)}(X_2) \\ & + (X_2 - \omega^{i_1})^2 q_{22}^{(i_1,i_2)}(X_2) \\ & + (X_1 - \omega^{i_1}) h_1'(\omega^{i_1}, \omega^{i_2}) + (X_2 - \omega^{i_2}) h_2'(\omega^{i_1}, \omega^{i_2}) + h(\omega^{i_1}, \omega^{i_2}) \end{aligned}$$

We now observe that the honestly computed opening proofs for (i_1, i_2), $\pi_{11}^{(i_1)}$, $\pi_{12}^{(i_1,i_2)}$ and $\pi_{22}^{(i_1,i_2)}$ are the evaluations of the quotient polynomials $q_{11}^{(i_1)}$, $q_{12}^{(i_1,i_2)}$ and $q_{22}^{(i_1,i_2)}$ at the secret point (τ_1, τ_2). Hence, the verification check is essentially checking the above equality at τ_1, τ_2, and will return one. □

Theorem A.1. *Let $h \in \mathsf{F}_p[X_1, X_2]$ be a bivariate polynomial. Then, for all $(u_1, u_2) \in \{0, \ldots, q-1\} \times \{0, \ldots, q-1\}$,*

$$h_1'(\omega^{u_1}, \omega^{u_2}) = q_1^{(u_1)}(\omega^{u_1}, \omega^{u_2}), h_2'(\omega^{u_1}, \omega^{u_2}) = q_2^{(u_1,u_2)}(\omega^{u_2}),$$

where

$$h(X_1, X_2) = (X_1 - \omega^{u_1}) q_1^{(u_1)}(X_1, X_2) + (X_2 - \omega^{u_2}) q_2^{(u_1,u_2)}(X_2) + h(\omega^{u_1}, \omega^{u_2}),$$

as defined in Lem. 4.1.

Proof. For any $u_2 \in \{0, \ldots, q-1\}$, consider the univariate polynomial $g_{u_2}(X_1) = h(X_1, \omega^{u_2})$. By putting $X_2 = \omega^{u_2}$ in Lem. 4.1, we know:

$$g_{u_2}(X_1) = (X_1 - \omega^{u_1}) q_1^{(u_1)}(X_1, \omega^{u_2}) + h(\omega^{u_1}, \omega^{u_2})$$

We can now apply [11, Thm.2] to get that:

$$g'_{u_2}(\omega^{u_1}) = q_1^{(u_1)}(\omega^{u_1}, \omega^{u_2}) \tag{8}$$

Next, observe that:

$$g_{u_2}(X_1) = \Sigma_{u\in\{0,\ldots,d\}} X_1^u \cdot h_u(\omega^{u_2})$$

Hence:

$$\begin{aligned} g'_{u_2}(X_1) &= \Sigma_{u\in\{0,\ldots,d\}} u \cdot X_1^{u-1} \cdot h_u(\omega^{u_2}) \\ &= h'_1(X_1, \omega^{u_2}) \end{aligned} \tag{9}$$

Combining Eqs. (8) and (9), we get that:

$$g'_{u_2}(\omega^{u_1}) = h'_1(\omega^{u_1}, \omega^{u_2}) = q_1^{(u_1)}(\omega^{u_1}, \omega^{u_2})$$

This proves the first part of the theorem.

Next, for any $u_1 \in \{0, \ldots, q-1\}$, consider a polynomial $r_{u_1}(X_2) = h(\omega^{u_1}, X_2)$. Again, by Lem. 4.1, we have:

$$r_{u_1}(X_2) = (X_2 - \omega^{u_2}) q_2^{(u_1,u_2)}(X_2) + h(\omega^{u_1}, \omega^{u_2})$$

We can now similarly apply [11, Thm.2] to get that:

$$r'_{u_1}(\omega^{u_2}) = q_2^{(u_1,u_2)}(\omega^{u_2})$$

Note that, $r'_{u_1}(X_2) = \Sigma_{u\in\{0,\ldots,d\}} \omega^{uu_1} \cdot h'_u(X_2) = h'_2(\omega^{u_1}, X_2)$, which implies that:

$$r'_{u_1}(\omega^{u_2}) = h'_2(\omega^{u_1}, \omega^{u_2})$$

Combining the last two equations, we prove that $h'_2(\omega^{u_2}) = q_2^{u_1,u_2}(\omega^{u_2})$. □

Proof of Thm. 4.3. We start with proving correctness of the opening proofs $\pi_{11}^{(i)}$.

Recall Eq. (4), which was obtained by applying Lem. A.3. We can define polynomials $G_1, \ldots, G_{d-1}$ as follows,

$$G_u(X_1, X_2) = \Sigma_{v\in\{u+1,\ldots,d\}} h_v(X_2) \cdot X_1^{v-u-1}$$

such that, $q_{11}^{(i)} = \Sigma_{u\in[d-1]} u \cdot G_u(X_1, X_2) \cdot (\omega^i)^{u-1}$.

Now, as mentioned in Sec. 4.2.2, we can compute all the opening proofs in three steps. For the first step, we claim that the vector $\boldsymbol{w}$ computed in lns. 7 to 10 of the BatchOpen algorithm contains the commitments to polynomials G_1 to G_{d-1}, specifically, $\boldsymbol{w}_i = [G_i(\tau_1, \tau_2)]_1$. To prove this, let us expand the $\boldsymbol{w}_i$'s:

$$\begin{aligned} \boldsymbol{w}_i = {} & h_{d,0} \cdot L_1[d-1-i, 0] + \ldots + h_{i+1,0} \cdot L_1[0,0] \\ & + h_{d,1} \cdot L_1[d-1-i, 1] + \ldots + h_{i+1,1} \cdot L_1[0,1] \\ & \vdots \\ & + h_{d,d-1} \cdot L_1[d-1-i, d-1] + \ldots + h_{i+1,d-1} \cdot L_1[0, d-1] \end{aligned}$$

We can re-arrange the terms and expand L_1 terms to get:

$$\begin{aligned} \boldsymbol{w}_i = {} & h_{d,0} \cdot [\tau_1^{d-1-i} \cdot \tau_2^0]_1 + \ldots + h_{d,d-1} \cdot [\tau_1^{d-1-u} \tau_2^{d-1}]_1 \\ & + h_{d-1,0} \cdot [\tau_1^{d-2-i} \cdot \tau_2^0]_1 + \ldots + h_{d-1,d-1} \cdot [\tau_1^{d-2-i} \tau_2^{d-1}]_1 \end{aligned}$$

$$
\begin{aligned}
&\vdots \\
&+ h_{i+1,0} \cdot [\tau_1^0 \tau_2^0]_1 + \ldots + h_{i+1,d-1} \cdot [\tau_1^0 \tau_2^{d-1}]_1 \\
&= [h_d(\tau_2) \cdot \tau_1^{d-1-i} + h_{d-1}(\tau_2) \cdot \tau_1^{d-2-i} + \ldots + h_{i+1}(\tau_2) \cdot \tau_1^0]_1 \\
&= [G_i(\tau_1, \tau_2)]_1
\end{aligned}
$$

Hence, $\boldsymbol{w}_i$ is indeed a commitment to G_i. Then, lns. 11 and 12 correspond exactly to steps (ii) and (iii) listed in Sec. 4.2.2, i.e. multiplying $\boldsymbol{w}_i$ by i for all $i \in [d-1]$, and then a Discrete Fourier transform over $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{d-1}$. Hence, the algorithm indeed computes the right openings $\pi_{11}^{(i)}$ for all $i \in \{0, \ldots, q-1\}$.

Next, for π_{22} proofs, recall Eq. (5). We start with proving that the $\hat{\pi}_2^{u,j}$ computed in lns. 14 to 19 do indeed correspond to commitments to quotients $\hat{q}_2^{(u,j)}(X_2)$. For that, let us focus on a particular value of $u \in \{0, \ldots, d\}$, i.e. $h_u(X_2)$. We define a bivariate polynomial of degree $\leq d$, $H^{(u)}(Y_1, Y_2) = h_u(Y_1)$. Then, we have:

$$H_0^{(u)}(Y_2) = h_{u,0}, \ldots, H_d^{(u)}(Y_2) = h_{u,d}$$

The quotients $\hat{q}_2^{(u,j)}(X_2)$ for $h_u(X_2)$ are equal to the quotients of $H^{(u)}$ with respect to $(Y_1 - \omega^j)^2$, by definition. We can now apply Lem. A.3 to $H^{(u)}$ to derive the following expression for the quotient $\hat{q}_2^{(u,j)}(X_2)$ for all $j \in \{0, \ldots, q-1\}$:

$$
\begin{aligned}
\hat{q}_2^{(u,j)}(X_2) = &\; h_{u,d} \cdot X_2^{d-2} + \\
&(h_{u,d-1} + 2\omega^j h_{u,d}) \cdot X_2^{d-3} + \\
&\vdots \\
&(h_{u,2} + 2\omega^j h_{u,3} + \ldots + (d-1)(\omega^j)^{d-2} h_{u,d}) \cdot X_2^0
\end{aligned}
$$

Re-arranging the above terms, we get:

$$
\begin{aligned}
\hat{q}_2^{(u,j)}(X_2) = &\\
&h_{u,d} X_2^{d-2} + \ldots + h_{u,2} \\
&2\omega^j (h_{u,d} X_2^{d-3} + \ldots + h_{u,3}) \\
&\vdots \\
&(d-1)(\omega^j)^{d-2} (h_{u,d} X_2^0)
\end{aligned}
$$

More succinctly, let us define $d-1$ polynomials $\hat{G}_{u,1}, \ldots, \hat{G}_{u,d-1}$ in $\mathsf{F}_p[X_2]$ such that:

$$\hat{q}_2^{(u,j)}(X_2) = \Sigma_{v \in [d-1]} v \cdot \hat{G}_{u,v}(X_2) \cdot (\omega^j)^{v-1} \tag{10}$$

It is easy to see that the $\boldsymbol{y}^{(u)}$ vector computed in ln. 15 of the BatchOpen algorithm (Fig. 12) contains commitments to $\hat{G}_{u,v}$ polynomials. Specifically, after ln. 15, we have:

$$\boldsymbol{y}_v^{(u)} = [\hat{G}_{u,v}(\tau_2)]_1 \forall v \in [d-1]$$

Next, in ln. 18, since we raise $\boldsymbol{y}_v^{(u)}$ to the power v, we get commitments to the polynomials $v \cdot \hat{G}_{u,v}(X_2)$. Formally, after ln. 18:

$$\boldsymbol{y}_v^{(u)} = [v \cdot \hat{G}_{u,v}(\tau_2)]_1 \forall v \in [d-1]$$

We claim that the terms $\hat{\pi}_{u,j}$ computed in ln. 19 are indeed commitments to $\hat{q}_2^{(u,j)}$ polynomials. To see this, observe that Eq. (10) implies that we can view $\hat{q}_2^{(u,j)}(\tau_2)$ as 'evaluations' of a degree $d-2$ polynomial with coefficients equal to $v \cdot \hat{G}_{u,v}(\tau_2)$ – hence the Discrete Fourier transform in ln. 19 gives us the correct commitments.

Lastly, observe that ln. 22 of the BatchOpen algorithm runs step (ii) mentioned above, i.e. a Discrete Fourier transform over the commitments to $\hat{q}_2^{(0,j)}, \ldots, \hat{q}_2^{(d,j)}$. Hence, we get the correct openings $\pi_{22}^{i,j}$ for all $i, j \in \{0, \ldots, q-1\}$.

Now, we move on to $\pi_{12}^{(i,j)}$. Recall Eq. (7), based on which we discussed a three-step procedure to compute these proofs. First, we observe that lns. 15 to 16 are simply applying the Feist–Khrovatovich technique [19] to polynomial h_u. Hence, by correctness of [19], we get that $\hat{\pi}^{u,j}$ is indeed equal to $[\hat{q}_1^{(u,j)}(\tau_2)]_1$ for all $u \in \{0, \ldots, d\}$ and all $j \in \{0, \ldots, q-1\}$. Next, ln. 17 executes Step (ii), wherein we scale the commitments $\hat{\pi}^{u,j}$ by u. Lastly, we claim that ln. 20 computes the correct opening proofs. To see this, observe that Eq. (7) implies that $q_{12}^{(i,j)}(\tau_2)$ can be seen as an 'evaluation' of a degree $d-2$ polynomial at ω^i with coefficients $\hat{q}_1^{1,j}(\tau_2), \ldots, d \cdot \hat{q}_1^{d,j}(\tau_2)$. ln. 20 executes a Discrete Fourier transform which gives us exactly these 'evaluations' as group elements. This completes the proof. □

Lemma A.2. *For a bivariate polynomial of degree d, i.e.* $h(X_1, X_2) = \Sigma_{u \in \{0,\ldots,d\}} h_u(X_2) \cdot X_1^u$, *we have that:*

$$q_1^{(i)}(X_1, X_2) = \frac{h(X_1,X_2) - h(\omega^i, X_2)}{X_1 - \omega^i} = h_d(X_2) \cdot X_1^{d-1} + (h_{d-1}(X_2) + \omega^i \cdot h_d(X_2)) \cdot X_1^{d-2} + \vdots (h_1(X_2) + \omega^i h_2(X_2) + \omega^{2i} h_3(X_2) + \ldots + \omega^{(d-1)i} h_d(X_2)) \cdot X_1^0$$

Proof. We prove the lemma by induction on the maximum degree of X_1, d_1. For any polynomial where the degree of X_1 is bounded by $d_1 = 0$, i.e. $h(X_1, X_2) = h_0(X_2)$ for some univariate polynomial $h_0(X_2)$, and $h_u(X_2) = 0$ for all $u \in [d]$. We know that the quotient $q_1^{(i)}(X_1, X_2) = 0$. The right-hand side also simplifies to zero, since $h_1, \ldots, h_d$ are all zero polynomials. Hence, our hypothesis is true for the base-case of $d_1 = 0$.

Let us assume it is true for polynomials where the degree of X_1 is bounded by some number d_1. We will now prove it is true for polynomials where the degree of X_1 is $d_1 + 1$. Let us use long division to divide $h(X_1, X_2)$ by $(X_1 - \omega^i)$. We will get the term $h_{d_1+1}(X_2) \cdot X_1^{d_1}$ in the quotient, and we get that:

$$q_1^{(i)} = h_{d_1+1}(X_2) \cdot X_1^{d_1} + \frac{\omega^i \cdot h_{d_1+1}(X_2) \cdot X_1^{d_1} + \Sigma_{j \in \{0,\ldots,d_1\}} h_j(X_2) \cdot X_1^j - h(\omega^i, X_2)}{X_1 - \omega^i}$$

Now, we have reduced the problem to computing the quotient for a different polynomial, whose degree in X_1 is bounded by d_1. We can now apply our induction hypothesis, and get the full expression – this proves the theorem. □

Lemma A.3. *For a bivariate polynomial* $h(X_1, X_2) = \Sigma_{u \in \{0,\ldots,d\}} h_u(X_2) \cdot X_1^u$, *and all* $i \in \{0, \ldots, q-1\}$ *we have that:*

$$q_{11}^{(i)}(X_1, X_2) = h_d(X_2) \cdot X_1^{d-2} + (h_{d-1}(X_2) + 2\omega^i h_d(X_2)) \cdot X_1^{d-3} + \vdots (h_2(X_2) + 2\omega^i h_3(X_2) + \ldots + (d-1)(\omega^i)^{d-2} h_d(X_2)) \cdot X_1^0$$

Proof. We can simply apply Eq. (6) twice. Specifically, we know that:

$$q_1^{(i)}(X_1, X_2) = \frac{h(X_1,X_2)-h(\omega^i,X_2)}{X_1-\omega^i} = h_d(X_2) \cdot X_1^{d-1} + \\ (h_{d-1}(X_2) + \omega^i \cdot h_d(X_2)) \cdot X_1^{d-2} + \\ \vdots \\ (h_1(X_2) + \omega^i h_2(X_2) + \omega^{2i} h_3(X_2) + \ldots + \omega^{(d-1)i} h_d(X_2)) \cdot X_1^0$$

Next, observe that $q_{11}^{(i)}(X_1, X_2)$ is just the result of dividing $q_1^{(i)}(X_1, X_2)$ by $(X_1 - \omega^i)$. Hence, applying Eq. (6) again, we get the theorem. □

Proof of Thm. 4.4. Let us define a modified game $\mathbf{G}'$, wherein, after the adversary sends the commitment com and a set of openings $(\hat{m}_j, \pi_j)$ for all positions j in J, the challenger runs CExt algorithm on com and the first opening – let us use m to denote the extracted message. The winning condition remains the same. Clearly, the adversary can win this game if and only if it wins the original game, i.e.:

$$\Pr[\mathbf{G'}_{\mathcal{A}} = 1] = \mathsf{Adv}^{\mathrm{loc-cor-open}}_{\mathcal{A},\mathsf{PSTMult}}(\lambda)$$

We now focus on bounding the advantage in this new game. We define two events:

- BadExtr. This is the event when $m = \perp$.
- BadPos. There exists $j \in J$ such that, $\hat{m}_j \neq \mathsf{C.Enc}(m)_j$.

By the law of total probability, we have:

$$\Pr[\mathbf{G'}_{\mathcal{A}} = 1] = \Pr[\mathbf{G'}_{\mathcal{A}} = 1 \wedge \mathsf{BadExtr}] + \\ \Pr[\mathbf{G'}_{\mathcal{A}} = 1 \wedge (\neg\mathsf{BadExtr} \wedge \mathsf{BadPos})] + \\ \Pr[\mathbf{G'}_{\mathcal{A}} = 1 \wedge (\neg\mathsf{BadExtr} \wedge \neg\mathsf{BadPos})]$$

We will now bound all three terms individually. We start with constructing an adversary $\mathcal{B}_1$ for extractability using $\mathcal{A}$. $\mathcal{B}_1$ gets ck from its challenger, and passes it on to $\mathcal{A}$. It then gets a commitment com, along with openings $\hat{m}_j, \pi_j$ for all positions $j \in J$ from $\mathcal{A}$. It forwards com, $j_1, \hat{m}_{j_1}, \pi_{j_1}$ to its challenger, where j_1 is the first element in J. It is easy to see that $\mathcal{B}_1$ wins its extractability game if $\mathcal{A}$ wins the game and if the event BadExtr occurs. Hence, we get:

$$\Pr[\mathbf{G'}_{\mathcal{A}} = 1 \wedge \mathsf{BadExtr}] = \mathsf{Adv}^{\mathrm{extr}}_{\mathcal{B}_1,\mathsf{PSTMult}}(\lambda)$$

Similarly, for the event $\neg$BadExtr $\wedge$ BadPos, we construct an adversary $\mathcal{B}_2$ which can break position-binding of PSTMult. It gets ck from its challenger, and passes it on to $\mathcal{A}$. It then gets a commitment com, along with openings $\hat{m}_j, \pi_j$ for all positions $j \in J$ from $\mathcal{A}$. It runs CExt on com, $j_1, \hat{m}_{j_1}, \pi_{j_1}$ to get a message m, and then runs $m' \leftarrow \mathsf{C.Enc}(m)$. It finds a position $j \in J$ such that $m'_j \neq \hat{m}_j$, and then sends com, $j, \hat{m}_j, \pi_j$ along with m'_j and $\pi'_j \leftarrow \mathsf{Open}(\mathsf{ck}, m, j)$ to its challenger. It is easy to see that if $\mathcal{A}$ wins its game and if the event $\neg$BadExtr $\wedge$ BadPos occurs, then (i) the proof π_j will pass verification, (ii) $m \neq \perp$, (iii) by correctness of PSTMult, the honestly-generated proof π'_j will also pass verification. Combining these with the fact that $\hat{m}_j \neq m'_j$, we get that $\mathcal{B}_2$ will win its position-binding game. Hence, we get that:

$$\Pr[\mathbf{G'}_{\mathcal{A}} = 1 \wedge (\neg\mathsf{BadExtr} \wedge \mathsf{BadPos})] = \mathsf{Adv}^{\mathrm{pos-bind}}_{\mathcal{B}_2,\mathsf{PSTMult}}(\lambda)$$

Lastly, we focus on the event $\neg$BadExtr $\wedge$ $\neg$BadPos. Here, we know that all openings $\hat{m}_j$ are consistent with the extracted message m. Since the opening proofs and the commitment is deterministic, this means all the opening proofs π_j must also equal the honestly-generated proofs, for all positions $j \in J$.

Now, we observe that if $\mathcal{A}$ wins its game, then the set J must have enough positions to recover the ith symbol of the encoding. This must mean that J has at least $d+1$ points on a line L passing through the point corresponding to position i i.e. ($i_1 = \lfloor i/q \rfloor, i_2 = i \mod q$). If this line is parallel to the y-axis, then we can trivially repair the proof $\pi_{11}^{i_1}$. Otherwise, we recall Sec. 4.2.3, wherein we discuss how the (honest) opening proofs π_{11} can be seen as 'evaluations' of a degree $d-2$ polynomial. Hence, the lagrange interpolation as seen in ln. 8 of the CLocalCorrect algorithm will indeed return the correct opening proof.

Similarly, for π_{12}, by Lem. A.4, we know that $\pi_{12}^{(i_1,i_2)}$ is equal to $[y(\omega^{i_1}, \omega^{i_2})]_1$ for a bivariate polynomial y of total degree $d-2$. Hence, we can use lagrange interpolation over any $d-1$ points on L, to obtain the proof for position i. This proves that the proof computed in ln. 14 is indeed valid. A similar argument holds for the proof π_{22} by relying on Lem. A.5, and observing the lagrange interpolation in ln. 15 of the CLocalCorrect algorithm. Hence, in the event $\neg\mathsf{BadExtr} \wedge \neg\mathsf{BadPos}$, the CLocalCorrect algorithm will successfully output valid opening proof for position i, meaning that:

$$\Pr[\mathbf{G'}_{\mathcal{A}} = 1 \wedge (\neg\mathsf{BadExtr} \wedge \neg\mathsf{BadPos})] = 0$$

This proves the theorem. □

Lemma A.4. *For a bivariate polynomial of degree d, i.e. $h(X_1, X_2) = \Sigma_{u\in\{0,\ldots,d\}} h_u(X_2) \cdot X_1^u$, let $\boldsymbol{y}^{(u)}$ be as defined in ln. 15 of Fig. 12, for all $u \in \{0,\ldots,d\}$. Let us denote $\boldsymbol{y}_i^{(u)} = [y_{u,i}]_1$ for all $u \in \{0,\ldots,d\}$ and all $i \in \{0,\ldots,d-1\}$. Then,*

(1) The polynomial $y(X_1, X_2) = \Sigma_{u\in[d], i\in\{0,\ldots,d-1\}} u \cdot y_{u,i} X_1^{u-1} X_2^i$ is a bivariate polynomial of total degree $\leq d-2$.
(2) The opening proof $\pi_{12}^{(i,j)}$ as computed in ln. 21 of Fig. 12, is equal to $= [y(\omega^i, \omega^j)]_1$ for all $i, j \in \{0,\ldots,q-1\}$.

Proof. For part one, we observe that the polynomial h has total degree d, so, for all $u \in \{0,\ldots,d\}$, the coefficients $h_{u,j}$ are zero for all $j > d-u$. This means that only the first $d-u$ rows are non-zero in the matrix in ln. 15 of Fig. 12. Hence, for each $u \in [d]$, the highest degree monomial is $X_1^{u-1} \cdot X_2^{d-u-1}$, i.e. total degree $d-2$. (Note that, for $u = d$, all the rows are just zeros – i.e. $y_{d,i} = 0 \; \forall \; i \in \{0,\ldots,d-1\}$)

For the second part, we first define univariate polynomials $y_u(X_2) = u \cdot y_{u,0} X_2^0 + \ldots + u \cdot y_{u,d-1} X_2^{d-1}$ for all $u \in [d]$. Then, we can write $y(X_1, X_2) = \Sigma_{u\in[d]} y_u(X_2) \cdot X_1^{u-1}$. Let us now trace how the opening proofs $\pi_{12}^{(i,j)}$ are computed.

First, observe that the terms $\hat{\pi}_1^{u,j}$ (see lns. 16 to 17 of Fig. 12) are computed by a Discrete Fourier transform on the vector $\boldsymbol{y}^{(u)}$ (and then scaled by u). This means that, for all $u \in [d]$ and all $j \in \{0, q-1\}$,

$$\hat{\pi}_1^{u,j} = [\Sigma_{i\in\{0,\ldots,d-1\}} u \cdot y_{u,i} \cdot (\omega^j)^i]_1$$

. This means, that $\hat{\pi}_1^{u,j}$ terms are the evaluations of the polynomial $y_u(X_2)$ at all roots of unity, in group $\mathbb{G}_1$, i.e. $\hat{\pi}_1^{u,j} = [y_u(\omega^j)]_1$ for all $j \in \{0,\ldots,d-1\}$.

Next, we observe that, for any $j \in \{0,\ldots,q-1\}$, the proofs $\pi_{12}^{(i,j)}$ for all i are computed via another Discrete fourier transform over $\{\hat{\pi}_1^{1,j}, \ldots, \hat{\pi}_1^{d,j}\}$ i.e. $\pi_{12}^{(i,j)} = [y_1(\omega^j) \cdot (\omega^i)^0 + y_2(\omega^j) \cdot (\omega^i)^1 \ldots + y_d(\omega^j) \cdot (\omega^i)^{d-1}]_1$, comparing this to the definition of the polynomial $y(X_1, X_2)$, we observe that this is exactly equal to the evaluation of this polynomial at the same point, i.e. $\pi_{12}^{(i,j)} = [y(\omega^i, \omega^j)]_1$. This proves the theorem. □

Lemma A.5. *For a bivariate polynomial of degree d, i.e. $h(X_1, X_2) = \Sigma_{u\in\{0,\ldots,d\}} h_u(X_2) \cdot X_1^u$, let $\boldsymbol{y}^{(u)}$ be as computed in ln. 18 of Fig. 12, for all $u \in \{0,\ldots,d\}$. Let us denote $\boldsymbol{y}_i^{(u)} = [y_{u,i}]_1$ for all $u \in \{0,\ldots,d\}$ and all $i \in \{0,\ldots,d-1\}$. Then,*

(1) The polynomial $y(X_1, X_2) = \Sigma_{u\in\{0,\ldots,d\}, i\in\{0,\ldots,d-2\}} y_{u,i+1} X_1^u X_2^i$ is a bivariate polynomial of total degree $\leq d-2$.
(2) The opening proof $\pi_{22}^{(i,j)}$ as computed in ln. 22 of Fig. 12, is equal to $= [y(\omega^i, \omega^j)]_1$ for all $i, j \in \{0,\ldots,q-1\}$.

Proof. For part one, similar to the proof of Lem. A.4, we observe that for all $u \in \{0, \ldots, d\}$, $h_{u,j}$ is zero for all $j > d - u$. Hence, only the first $d - u$ rows are non-zero in the matrix-vector product computed in ln. 15. Hence, for each $u \in \{0, \ldots, d\}$, the highest degree monomial in y is $X_1^u \cdot X_2^{d-u-2}$, i.e. degree $d - 2$.

For the second part, we first define univariate polynomials $y_u(X_2) = y_{u,1}X_2^0 + y_{u,2}X_2^1 + \ldots + y_{u,d-1}X_2^{d-2}$, for all $u \in \{0, \ldots, d\}$. Now, let us trace how the proofs $\pi_{22}^{(i,j)}$ are computed. First, observe that the terms $\hat{\pi}_2^{(u,j)}$ (see ln. 19) are computed by a Discrete fourier transform on $\boldsymbol{y}_1^{(u)}, \ldots, \boldsymbol{y}_{d-1}^{(u)}$. This means that, for all $u \in \{0, \ldots, d\}$ and all $j \in \{0, \ldots, q-1\}$:

$$\hat{\pi}_2^{(u,j)} = [\Sigma_{i \in \{0,\ldots,d-2\}} y_{u,i+1} \cdot (\omega^j)^i]_1 = [y_u(\omega^j)]_1$$

Next, we observe that, for any $j \in \{0, \ldots, q-1\}$, the proofs $\pi_{22}^{(i,j)}$ for all i are computed via another Discrete fourier transform, as in ln. 22. This implies that:

$$\pi_{22}^{(i,j)} = \Sigma_{v \in \{0,\ldots,d\}} (\omega^i)^v \cdot [y_v(\omega^j)]_1$$

Comparing this to the definition of $y(X_1, X_2)$, we see that the above is exactly equal to $[y(\omega^i, \omega^j)]_1$. This proves the theorem. □

Proof of Thm. 4.6. Let $\mathrm{G}_{\mathsf{PSTMult}}^{pos-bind,(0)}$ denote the original position-binding game. We first define a slightly different game, $\mathrm{G}_{\mathsf{PSTMult}}^{pos-bind,(1)}$ wherein the challenger samples the commitment key ck as follows. Specifically, it first guesses the position $i \in \{0, \ldots, q^2 - 1\}$ for which $\mathcal{A}$ will break position-binding. It defines $i_1 \leftarrow \lfloor i/q \rfloor$ and $i_2 \leftarrow i \mod q$. It then samples τ_1 uniformly randomly – exactly the same as in the original game. But now, it samples random $r, s \in \mathsf{F}_p$ such that, $\omega^{i_2} = r \cdot \omega^{i_1} + s$. It then sets $\tau_2 \leftarrow r \cdot \tau_1 + s$. It uses τ_1 and τ_2 to compute all the terms in L_1 and L_2, in the same way as in the original game. We note that the view of the adversary is identically distributed in both the games, since r, s are sampled uniformly randomly – so τ_2 is still distributed uniformly at random. The adversary then responds with the position i^*, along with two valid openings and proofs. The challenger aborts if $i \neq i^*$. The rest of the game remains the same.

Hence, we get that the new game outputs 1 if the original game outputs 1 and if the challenger guesses the position i correctly. More formally:

$$\frac{1}{q^2} \cdot \Pr[\mathrm{G}_{\mathsf{PSTMult},\mathcal{A}}^{pos-bind,(0)} = 1] = \Pr[\mathrm{G}_{\mathsf{PSTMult},\mathcal{A}}^{pos-bind,(1)} = 1]$$

We will now focus on bounding the advantage of $\mathcal{A}$ in the new game. Let us use (z, z_1, z_2), $(\pi_{11}, \pi_{12}, \pi_{22})$ and (z', z_1', z_2') and $(\pi_{11}', \pi_{12}', \pi_{22}')$ to denote the two openings and the corresponding proofs output by $\mathcal{A}$. Since both the proofs successfully verify, we know that,

$$\begin{gathered} e(\mathsf{com} - [z]_1 - [z_1(\tau_1 - \omega^{i_1})]_1 - [z_2(\tau_2 - \omega^{i_2})], [1]_2) = \\ e(\pi_{11}, [(\tau_1 - \omega^{i_1})^2]_2) + \\ e(\pi_{12}, [(\tau_1 - \omega^{i_1})(\tau_2 - \omega^{i_2})]_2) + \\ e(\pi_{22}, [(\tau_2 - \omega^{i_2})^2]_2) \end{gathered} \tag{11}$$

And similarly,

$$\begin{gathered} e(\mathsf{com} - [z']_1 - [z_1'(\tau_1 - \omega^{i_1})]_1 - [z_2'(\tau_2 - \omega^{i_2})], [1]_2) = \\ e(\pi_{11}', [(\tau_1 - \omega^{i_1})^2]_2) + \\ e(\pi_{12}', [(\tau_1 - \omega^{i_1})(\tau_2 - \omega^{i_2})]_2) + \\ e(\pi_{22}', [(\tau_2 - \omega^{i_2})^2]_2) \end{gathered} \tag{12}$$

We now consider three different events: let E_1 denote the event that $z \neq z'$. Let E_2 denote the event that $z = z'$ but $(z'_1 - z_1) + r(z'_2 - z_2) \neq 0$. Lastly, let E_3 denote the event that $z = z'$ and $(z'_1 - z_1) + r(z'_2 - z_2) = 0$. Clearly,

$$\begin{aligned}\Pr[\mathbf{G}^{pos-bind,(1)}_{\mathsf{PSTMult},\mathcal{A}} = 1] = \Pr[\mathbf{G}^{pos-bind,(1)}_{\mathsf{PSTMult},\mathcal{A}} = 1 \wedge \mathsf{E}_1] \\ + \Pr[\mathbf{G}^{pos-bind,(1)}_{\mathsf{PSTMult},\mathcal{A}} = 1 \wedge \mathsf{E}_2] + \Pr[\mathbf{G}^{pos-bind,(1)}_{\mathsf{PSTMult},\mathcal{A}} = 1 \wedge \mathsf{E}_3]\end{aligned}$$

We now bound the three probabilities separately.

We will start with the first case, i.e. event E_1. We construct an adversary $\mathcal{B}_1$ for the d-SBDH game, using $\mathcal{A}$. Specifically, $\mathcal{B}_1$ acts as the challenger to $\mathcal{A}$ in the position-binding game $pos - bind, (1)$, as follows:

- $\mathcal{B}_1$ receives $e, p, \{[\tau^0]_1, [\tau]_1, \ldots, [\tau^\ell]_1\}$ and $\{[\tau^0]_2, [\tau]_2, \ldots, [\tau^\ell]_2\}$ from its d-SBDH challenger.
- $\mathcal{B}_1$ guesses the position i and computes $i_1 \leftarrow \lfloor i/q \rfloor$ and $i_2 \leftarrow i \mod q$.
- $\mathcal{B}$ generates the commitment key as follows: it sets $\tau_1 = \tau$. Note that $\mathcal{B}$ does not know this value, but we will show that it can still generate the key.
- $\mathcal{B}$ samples random $r, s \in \mathsf{F}_p$ as described above. It then sets $\tau_2 \leftarrow r \cdot \tau + s$. Again, note that $\mathcal{B}$ does not know this value.
- Observe that the commitment key needs to include the following term for all $i, j \geq 0$ such that $i + j \leq d$:

  $$L_1[(i,j)] = [\tau_1^i \tau^j]_1 = [\tau^i \cdot (r\tau + s)^j]_1.$$

  Since $\mathcal{B}_1$ has $[\tau^u]_1$ for $u \in \{0, \ldots, d\}$, it can compute $L_1[i,j]$ as a linear combination of these terms. Similarly, it can compute $L_2[(i,j)]$ for all $i, j \geq 0, i + j \in \{1, 2\}$ by taking linear combinations over $\{[\tau^u]_2\}_{u \in \{0,\ldots,2\}}$. $\mathcal{B}$ then sends $\mathsf{ck} \leftarrow (d, q, L_1, L_2)$ to $\mathcal{A}$.

$\mathcal{B}_1$ then receives from $\mathcal{A}$ two valid evaluations and opening proofs for position i'. It aborts if $i' \neq i$, as in the game. Observe that $\mathcal{B}_1$ perfectly simulates the game $\mathbf{G}^{pos-bind,(1)}$ to $\mathcal{A}$.

We now make the following observations to show how $\mathcal{B}_1$ can win the d-SBDH game if $\mathcal{A}$ wins its game and if E_1 occurs. We first subtract Eq. (11) from Eq. (12) and re-arrange some terms to get,

$$\begin{gathered}e([z - z']_1, [1]_2) = \\ e(\pi'_{11} - \pi_{11}, [(\tau_1 - \omega^{i_1})^2]_2) + \\ e(\pi'_{12} - \pi_{12}, [(\tau_1 - \omega^{i_1})(\tau_2 - \omega^{i_2})]_2) + \\ e(\pi'_{22} - \pi_{22}, [(\tau_2 - \omega^{i_2})^2]_2) + \\ e([(z'_1 - z_1)(\tau_1 - \omega^{i_1})]_1 + [(z'_2 - z_2)(\tau_2 - \omega^{i_2})]_1, [1]_2)\end{gathered}$$

We now simplify the above by observing that $\tau_2 - \omega^{i_2} = r(\tau_1 - \omega^{i_1})$:

$$\begin{gathered}e([z - z']_1, [1]_2) = \\ e(\pi'_{11} - \pi_{11}, [(\tau_1 - \omega^{i_1})^2]_2) + \\ e(\pi'_{12} - \pi_{12}, [r(\tau_1 - \omega^{i_1})^2]_2) + \\ e(\pi'_{22} - \pi_{22}, [r^2(\tau_1 - \omega^{i_1})^2]_2) + \\ e([(z'_1 - z_1)(\tau_1 - \omega^{i_1})]_1 + [r(z'_2 - z_2)(\tau_1 - \omega^{i_1})]_1, [1]_2)\end{gathered}$$

Note that all of the terms on the right have the term $(\tau_1 - \omega^{i_1})$. Hence, $\mathcal{B}$ can compute $(\tau_1 - \omega^{i_1})^{-1} \cdot e([1]_1, [1]_2)$ by removing the $(\tau_1 - \omega^{i_1})$ factor from each term on the right side, and then raising it to power $(z - z')^{-1}$. Specifically, $\mathcal{B}_!$

computes the following:

$$y = (z - z')^{-1} \cdot \begin{pmatrix} e(\pi'_{11} - \pi_{11}, [(\tau_1 - \omega^{i_1})]_2) + \\ e(\pi'_{12} - \pi_{12}, [r(\tau_1 - \omega^{i_1})]_2) + \\ e(\pi'_{22} - \pi_{22}, [r^2(\tau_1 - \omega^{i_1})]_2) + \\ e([(z'_1 - z_1)]_1 + [r(z'_2 - z_2)]_1, [1]_2) \end{pmatrix}$$

$\mathcal{B}_1$ then returns $(-\omega^{i_1}, y)$ to its challenger. Based on the equations above, we know that this is indeed a valid d-SBDH tuple, hence, $\mathcal{B}_1$ will win its game if (i) $\mathcal{A}$ wins its game and gives two valid openings and proofs, and (ii) event E_1 occurs. More formally:

$$\Pr[\mathbf{G}^{pos-bind,(1)}_{\mathcal{A},\mathsf{PSTMult}} = 1 \wedge \mathsf{E}_1] = \mathsf{Adv}^{\mathrm{d-sbdh}}_{\mathcal{B}_1}(\lambda)$$

Let us now consider the second case, i.e. event E_2, where $z = z'$ but $(z'_1 - z_1) + r(z'_2 - z_2) \neq 0$. We now construct an adversary $\mathcal{B}_2$ for the d-SBDH game, similar to $\mathcal{B}_1$. Specifically, it receives the terms $e, p, \{[\tau^0]_1, [\tau]_1, \ldots, [\tau^\ell]_1\}$ and $\{[\tau^0]_2, [\tau]_2, \ldots, [\tau^\ell]_2\}$ from its d-SBDH challenger. It guesses i, computes i_1, i_2, sets $\tau_1 = \tau$, and samples r, s, τ_2 in the exact manner as $\mathcal{B}_1$, i.e. $\tau_2 = r \cdot \tau + s$. It then computes the ck in the same way, and sends it to $\mathcal{A}$. Lastly, it receives two valid openings and proofs from $\mathcal{A}$.

In this case, we can again subtract Eq. (11) from Eq. (12) and re-arrange some terms to get:

$$\begin{gathered} e([(z_1 - z'_1)(\tau_1 - \omega^{i_1})]_1 + [(z_2 - z'_2)(\tau_2 - \omega^{i_2})]_1, [1]_2) = \\ e(\pi'_{11} - \pi_{11}, [(\tau_1 - \omega^{i_1})^2]_2) + \\ e(\pi'_{12} - \pi_{12}, [(\tau_1 - \omega^{i_1})(\tau_2 - \omega^{i_2})]_2) + \\ e(\pi'_{22} - \pi_{22}, [(\tau_2 - \omega^{i_2})^2]_2) \end{gathered}$$

We can simplify this further by again using the fact that $\tau_2 - \omega^{i_2} = r(\tau_1 - \omega^{i_1})$:

$$\begin{gathered} e([((z_1 - z'_1) + r((z_2 - z'_2))) \cdot (\tau_1 - \omega^{i_1})]_1, [1]_2) = \\ e(\pi'_{11} - \pi_{11}, [(\tau_1 - \omega^{i_1})^2]_2) + \\ e(\pi'_{12} - \pi_{12}, [r(\tau_1 - \omega^{i_1})^2]_2) + \\ e(\pi'_{22} - \pi_{22}, [r^2(\tau_1 - \omega^{i_1})^2]_2) \end{gathered}$$

Let $\mathsf{E}_{2,\mathrm{b}}$ denote the event that $\tau_1 = \omega^{i_1}$ – note that this happens only with probability $1/p$ since $\tau_1 = \tau$, which is uniformly randomly sampled by the d-SBDH challenger. We claim that in the event where $\mathcal{A}$ wins its game, and $\mathsf{E}_2 \wedge \neg\mathsf{E}_{2,\mathrm{b}}$, $\mathcal{B}$ can win its d-SBDH game. Specifically, $\mathcal{B}_2$ can compute

$$y = ((z_1 - z'_1) + r((z_2 - z'_2)))^{-1} \cdot \begin{pmatrix} e(\pi'_{11} - \pi_{11}, [1]_2) + \\ e(\pi'_{12} - \pi_{12}, [r]_2) + \\ e(\pi'_{22} - \pi_{22}, [r^2]_2) \end{pmatrix}$$

$\mathcal{B}_2$ then sends $(-\omega^{i_1}, y)$ to its challenger. Based on the equations above, this is indeed a valid d-SBDH solution. Hence, we get that,

$$\Pr[\mathbf{G}^{pos-bind,(1)}_{\mathcal{A},\mathsf{PSTMult}} = 1 \wedge \mathsf{E}_2 \wedge \neg\mathsf{E}_{2,\mathrm{b}}] = \mathsf{Adv}^{\mathrm{d-sbdh}}_{\mathcal{B}_2}(\lambda)$$

This implies:

$$\Pr[\mathbf{G}^{pos-bind,(1)}_{\mathcal{A},\mathsf{PSTMult}} = 1 \wedge \mathsf{E}_2] - 1/p \leq \mathsf{Adv}^{\mathrm{d-sbdh}}_{\mathcal{B}_2}(\lambda)$$

We now move on to the last case, wherein $z = z'$ and $(z'_1 - z_1) + r(z'_2 - z_2) = 0$. We again construct an adversary $\mathcal{B}_3$ for the d-SBDH game. $\mathcal{B}_3$ receives the terms $e, p, \{[\tau^0]_1, [\tau]_1, \ldots, [\tau^\ell]_1\}$ and $\{[\tau^0]_2, [\tau]_2, \ldots, [\tau^\ell]_2\}$ from its d-SBDH

challenger. It computes i_1, i_2 as in the game, and samples τ_1 uniformly at random. Now, it sets r to be τ. Hence, $s = \omega^{i_2} - \tau \cdot \omega^{i_1}$ and $\tau_2 = \tau\tau_1 + s$, which is $\tau(\tau_1 - \omega^{i_1}) + \omega^{i_2}$. Note that $\mathcal{B}_3$ does not know these values since it does not know τ. Now, $\mathcal{B}_3$ needs to compute the commitment key. Observe that:

$$L_1[(i,j)] = [\tau_1^i \tau_2^j]_1 = [\tau_1^i (\tau(\tau_1 - \omega^{i_1}) + \omega^{i_2})^j]$$

Since $\mathcal{B}_3$ has $\{[\tau^u]_1\}_{u\in\{0,d\}}$, it can take linear combinations over these terms to compute $L_1[(i,j)]$ for all $i, j \geq 0, i+j \leq d$. Similarly, it can use $\{[\tau^u]_2\}_{u\in\{0,d\}}$ to compute all the terms in L_2. It then sends $\mathsf{ck} = (d, q, L_1, L_2)$ to $\mathcal{A}$, and gets two valid openings and proofs for a position i'. It aborts if $i' \neq i$.

In the event E_3, we know that $z = z'$. Hence, (i) either $z_1 \neq z_1'$ or $z_2 \neq z_2'$ (or both). Moreover, we know that (ii) $(z_1' - z_1) + r(z_2' - z_2) = 0$. Combining (i) and (ii), we get that $z_1 \neq z_1'$ and $z_2 \neq z_2'$. Hence, $\mathcal{B}_3$ can directly compute r as $\frac{z_1 - z_1'}{z_2' - z_2}$. It can then use this to easily break d-SBDH, e.g. by sampling $c \leftarrow\!\!\$\ \mathsf{F}_p$, and computing $\frac{1}{r+c} \cdot e([1]_1, [1]_2)$. Hence, we get that:

$$\Pr[\mathbf{G}^{pos-bind,(1)}_{\mathcal{A},\mathsf{PSTMult}} = 1 \wedge \mathsf{E}_3] = \mathsf{Adv}^{\mathrm{d-sbdh}}_{\mathcal{B}_3}(\lambda)$$

This completes the proof. □

Proof of Thm. 4.7. Let us consider $\mathcal{A}$ playing the extractability game in the AGM. It gets as input the commitment key ck, which contains group elements of the form $L_1[(i,j)] = [\tau_1^i \cdot \tau_2^j]_1$ for all $i, j \geq 0$ such that $i + j \leq d$ and $L_2[(i,j)] = [\tau_1^i \cdot \tau_2^j]_2$ for all $i, j \geq 0$ such that $1 \leq i + j \leq 2$.

$\mathcal{A}$ then outputs a commitment $\mathsf{com} \in \mathbb{G}_1$, along with an opening proof for a position i, i.e. $\hat{m}, \pi$. Since we're in the AGM, $\mathcal{A}$ must output coefficients $\alpha_{i,j}$ for all $i, j \geq 0, i + j \leq d$ such that $\mathsf{com} = \Sigma_{i,j\geq 0, i+j\leq d} \alpha_{i,j} \cdot L_1[(i,j)]$. Define a bivariate polynomial $\hat{h}(X_1, X_2) = \Sigma_{i,j\geq 0, i+j\leq d} \alpha_{i,j} \cdot X_1^i X_2^j$. We claim that this is indeed the committed message.

To prove that, let us try to commit to this polynomial. Since Commit simply takes a linear combination of the terms in L_1 with coefficients of h, it will be exactly equal to com. □

B Soundness proofs for DAS compiler

Proof of Thm. 3.10. First, observe that even if the adversary corrupts all the n storage nodes, since the dispersal was done honestly, the sampling clients will be able to successfully retrieve the encoding symbols and their proofs from the network. Since the encoding and the proofs are honestly generated, the probability that any of the proofs do not verify is bounded by the correctness error of the commitment scheme, which is assumed to be negligible. □

Proof of Thm. 3.11. Consider an adversary $\mathcal{A}$ playing the worst-case soundness game. We start with some notation. Let com be the commitment, and n be the number of parties returned by $\mathcal{A}$. Let Δ denote the distance of the code C, which means that it is possible to decode the message from any $1 - \Delta$ fraction of the codeword symbols. Let $\mathcal{T} \subseteq [\ell]$ denote the set of verifiors whose queries were successfully answered by $\mathcal{A}$. We know that for $\mathcal{A}$ to win, $|\mathcal{T}| \geq \delta\ell$. Let $\mathsf{tran}_i = \{(j_{i,k}, \hat{m}_{j_{i,k}}, \pi_{j_{i,k}})\}_{k\in[Q]}$ for all $i \in \mathcal{T}$. For each $i \in \mathcal{T}$, let $\mathcal{I}_i$ be the set of codeword positions contained in the ith transcript, such that the opening proof is valid, i.e. $\mathcal{I}_i = \{j : \exists k \in [Q] \text{ s.t. } j_{i,k} = j \wedge \mathsf{CVerify}(\mathsf{ck}, \mathsf{com}, j_{i,k}, \hat{m}_{j_{i,k}}, \pi_{j_{i,k}}) = 1\}$.

We observe that there are two ways for the adversary to win this game. First, if there exists a subset of the ℓ verifiors of size $\ell\delta$ such that all of the samples of all of these verifiers fall within a $(1 - \Delta)$ fraction of the indices, then, these samples are not enough to decode the original data, even if the data was dispersed honestly by $\mathcal{A}$, and hence extraction would not succeed. Let E_{ne} denote this event. Second, if the adversary breaks some notion of binding for the commitment

scheme, then again decoding would not succeed, even if sampling was successful. Specifically, we define the following events:

- E_{pb}: This event occurs if there is an index j in two transcripts, such that the claimed jth symbol is different in the two transcripts but the opening proof is valid for both.
- $\mathsf{E}_{\mathsf{inv}}$: This event occurs if Dec outputs $\perp$ because there are enough codeword symbols with valid proofs in the transcripts, but they do not correspond to any codeword.
- $\mathsf{E}_{\mathsf{inc}}$: This event occurs if Dec outputs some data data′, but its commitment is not equal to com.

Let E denote the event $\mathsf{E}_{\mathsf{ne}} \vee \mathsf{E}_{\mathsf{pb}} \vee \mathsf{E}_{\mathsf{inv}} \vee \mathsf{E}_{\mathsf{inc}}$.
We have:

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{A},}(\lambda) &= \Pr[\mathbf{G} = 1] \\
&= \Pr[\mathbf{G} = 1|\mathsf{E}] \cdot \Pr[\mathsf{E}] + \Pr[\mathbf{G} = 1|\neg\mathsf{E}] \cdot \Pr[\neg\mathsf{E}] \\
&\leq \Pr[\mathsf{E}] \qquad (13) \\
&\leq \Pr[\mathsf{E}_{\mathsf{pb}}] + \Pr[\mathsf{E}_{\mathsf{inv}}] + \Pr[\mathsf{E}_{\mathsf{inc}}] + \Pr[\mathsf{E}_{\mathsf{ne}}] \qquad (14)
\end{aligned}$$

Equation (13) follows from the fact that $\Pr[\mathbf{G} = 1|\neg\mathsf{E}] = 0$ because in this event, any subset of $\delta\ell$ transcripts contains enough positions, moreover, the Dec algorithm of the code successfully finds a message m such that $\mathsf{Commit}(m) = \mathsf{com}$. Equation (14) follows from the union bound. We will now bound these terms separately. First, for the event E_{ne}, the probability that all of the samples of a particular subset of $\ell\delta$ transcripts fall into a $(1-\Delta)$ fraction of the codeword is bounded by:

$$\binom{M}{(1-\Delta)M} \cdot (1-\Delta)^{\ell\delta Q}$$

Next, we use a union bound over all possible subsets of size $\ell\delta$, to get that:

$$\Pr[\mathsf{E}_{\mathsf{ne}}] \leq \binom{\ell}{\ell\delta} \cdot \binom{\mathsf{M}}{(1-\Delta)\mathsf{M}} \cdot (1-\Delta)^{\ell\delta \mathsf{Q}}$$

This is similar to the global safety analysis in [26]. Note that the union bound over all subsets of size $\ell\delta$ is what was missing in the analysis of [31].

Next, we construct an adversary $\mathcal{B}_1$ that can break position binding in the event E_{pb}. More formally, $\mathcal{B}$ acts as the challenger in the worst-case soundness game. It receives the commitment key ck from its challenger, which it passes to $\mathcal{A}$ as public parameters. It then gets n, com along with C from $\mathcal{A}$. It honestly runs the Verify algorithm ℓ times. It then checks if there exists a position $j \in [M]$ in any two transcripts $(j, \hat{m}_j, \pi_j) \in t_{i_1}$ and $(j, \hat{m}'_j, \pi'_j) \in t_{i_2}$ for some $i_1 \neq i_2 \in [\ell]$ such that $\hat{m}_j \neq \hat{m}'_j$. If so, it responds to its challenger with $(\mathsf{com}, j, \hat{m}_j, \pi_j, \hat{m}'_j, \pi'_j)$. It is easy to see that $\mathcal{B}_1$ perfectly simulates the worst-case soundness game, and $\mathcal{B}_1$ wins its position-binding game in the event E_{pb}. Hence, we get that:

$$\mathsf{Adv}^{\mathsf{pos-bind}}_{\mathcal{B}_1,\mathsf{Com}}(\lambda) = \Pr[\mathsf{E}_{\mathsf{pb}}]$$

Similarly, we construct an adversary $\mathcal{B}_2$ that can break code-binding in the event $\mathsf{E}_{\mathsf{inv}}$. More formally, $\mathcal{B}_2$ simulates the worst-case soundness game to $\mathcal{A}$. Similar to $\mathcal{B}_1$, it receives ck from its challenger, and sends it to $\mathcal{A}$. It then gets n, com from $\mathcal{A}$. It honestly runs the Verify algorithm ℓ times, and then tries to run the Ext on the transcripts. If the total number of unique positions across all transcripts is $> (1-\Delta)M$ but Dec outputs $\perp$, then $\mathcal{B}_2$ sends com alongwith all the transcripts to its challenger. Otherwise it aborts. It can be seen that $\mathcal{B}_2$ perfectly simulates the worst-case soundness

game, and $\mathcal{B}_2$ wins its code-binding game in the event $\mathsf{E}_{\mathsf{inv}}$. Hence, we get that:

$$\mathsf{Adv}^{\mathsf{code-bind}}_{\mathcal{B}_2,\mathsf{Com}}(\lambda) = \Pr[\mathsf{E}_{\mathsf{inv}}]$$

Lastly, we construct $\mathcal{B}_3$ that breaks reconstruction-binding in the event $\mathsf{E}_{\mathsf{inc}}$. $\mathcal{B}_3$ simulates the worst-case soundness game to $\mathcal{A}$ as before. On getting the commitment com from $\mathcal{A}$, it runs the CExt algorithm to extract the message m such that $\mathsf{Commit}(m) = \mathsf{com}$. Then, it runs the Dec algorithm to get another message m'. It then computes the encoding $\hat{m}'$ of m', i.e. $\hat{m}' = \mathsf{Enc}(m')$, and an opening proof for all the positions, i.e. $\pi'_i = \mathsf{Open}(\mathsf{ck}, m', i)$ for all $i \in [M]$. It sends to its challenger, (i) the commitment com, (ii) the encoding $\hat{m}'$ and its opening proofs $\{\pi'_i\}$ and (iii) the list of positions and the corresponding symbols and opening proofs from the verification transcripts. $\mathcal{B}_3$ perfectly simulates the worst-case soundness game. Let $\mathsf{E}_{\mathsf{ext}}$ denote the event that the extraction algorithm CExt does not succeed. Observe that $\mathcal{B}_3$ wins its reconstruction-binding game in the event $\mathsf{E}_{\mathsf{inc}}$ and $\neg\mathsf{E}_{\mathsf{ext}}$. More formally:

$$\mathsf{Adv}^{\mathsf{rec-bind}}_{\mathcal{B}_3,\mathsf{Com}}(\lambda) = \Pr[\mathsf{E}_{\mathsf{inc}} \wedge \neg\mathsf{E}_{\mathsf{ext}}] \tag{15}$$

$$\geq \Pr[\mathsf{E}_{\mathsf{inc}}] - \Pr[\mathsf{E}_{\mathsf{ext}}] \tag{16}$$

Additionally, we observe that if the extraction does not succeed, then we can use $\mathcal{B}_3$ to make another adversary $\mathcal{B}_4$ which wins the extractability game with probability $\Pr[\mathsf{E}_{\mathsf{ext}}]$. Hence, we get that:

$$\Pr[\mathsf{E}_{\mathsf{inc}}] \leq \mathsf{Adv}^{\mathsf{rec-bind}}_{\mathcal{B}_3,\mathsf{Com}}(\lambda) + \mathsf{Adv}^{\mathsf{extr}}_{\mathcal{B}_4,\mathsf{Com}}(\lambda)$$

Combining all the inequalities proves the theorem. □

Proof of Thm. 3.12. Consider an adversary playing the optimistic-case soundness game. We start with some notation. Let com be the commitment, and n be the number of parties returned by $\mathcal{A}$. Let $\mathcal{T} \subseteq [\ell]$ denote the set of verifiors whose queries were successfully answered by $\mathcal{A}$. We know that for $\mathcal{A}$ to win, $|\mathcal{T}| \geq \delta\ell$. Let $\mathsf{tran}_i = \{(j_{i,k}, \hat{m}_{j_{i,k}}, \pi_{j_{i,k}})\}_{k\in[Q]}$ for all $i \in \mathcal{T}$. For each $i \in \mathcal{T}$, let $\mathcal{I}_i$ be the set of codeword positions contained in the ith transcript, such that the opening proof is valid, i.e. $\mathcal{I}_i = \{j : \exists k \in [Q] \text{ s.t. } j_{i,k} = j \wedge \mathsf{CVerify}(\mathsf{ck}, \mathsf{com}, j_{i,k}, \hat{m}_{j_{i,k}}, \pi_{j_{i,k}}) = 1\}$.

We observe that there are two ways for the adversary to win this game. First, if the adversary breaks some notion of binding for the commitment scheme, then decoding would not succeed, even if enough verifiors think that the data is available. Specifically, similar to the proof of Thm. 3.11, we define the events E_{pb}, $\mathsf{E}_{\mathsf{inv}}$ and $\mathsf{E}_{\mathsf{inc}}$.

Second, if there exists a subset of the ℓ verifiors of size $\ell\delta$ such that all of the samples queried by these verifiors fall within a $1 - \Delta + f$ fraction of the indices, then, the adversary can (i) disperse only these samples, (ii) corrupt any f fraction of nodes storing any fM of these symbols, and now, the honest nodes can only store the remaining $(1 - \Delta)M$ samples, which are not enough to decode. Let $\mathsf{E}_{\mathsf{ne,o}}$ denote this event.

First, we argue that in the event $\mathsf{E} = \neg\mathsf{E}_{\mathsf{ne,o}} \wedge \neg\mathsf{E}_{\mathsf{pb}} \wedge \neg\mathsf{E}_{\mathsf{inv}} \wedge \neg\mathsf{E}_{\mathsf{inc}}$, the advantage of the adversary is zero. Specifically, all the dispersed samples do indeed have valid opening proofs, and are consistent with a particular message m. Moreover, for any subset S of $\delta\ell$ verifiors whose queries the adversary chooses to answer, their total number of samples are more than $(1 - \Delta + f)M$. Hence, even if the adversary corrupts f fraction of nodes (making fM of these symbols unavailable to honest nodes), the honest nodes will still have more than $(1 - \Delta)M$ samples, which are now enough to decode. Note that this holds because of network assumption, that if the verifiors are able to receive a particular sample, then the honest nodes who are supposed to store this sample must also have received it.

Hence, we just need to bound the probability of the event $\neg\mathsf{E}$ to bound the advantage of the adversary.

Similar to Thm. 3.11, we can construct an adversary $\mathcal{B}_1$ which can break position-binding if $\mathcal{A}$ wins the optimistic-case soundness game and the event E_{pb} occurs. Hence, we have:

$$\mathsf{Adv}^{\mathsf{pos-bind}}_{\mathcal{B}_1,\mathsf{Com}}(\lambda) = \Pr[\mathsf{E}_{\mathsf{pb}}]$$

Similarly, we can construct an adversary $\mathcal{B}_2$ that can break code-binding in the event $\mathsf{E}_{\mathsf{inv}}$. $\mathcal{B}_2$ simulates the optimistic-case soundness game to $\mathcal{A}$. It receives ck from its challenger and sends it to $\mathcal{A}$. It then gets n, com from $\mathcal{A}$. It honestly runs the Verify algorithm ℓ times. It then updates chunks of honest nodes, d_i based on the samples received by the verifiors (this is reasonable again because of our network assumption). It tries to run Retrieve protocol with all the nodes. If the total number of unique positions across the transcripts is $> (1-\Delta)M$ but the Dec algorithm (run by Retrieve) outputs $\perp$, then $\mathcal{B}_2$ sends com along with all the samples and their proofs to its challenger. It can be seen that $\mathcal{B}_2$ perfectly simulates the optimistic-case soundness game, and it wins the code-binding game if event $\mathsf{E}_{\mathsf{inv}}$ occurs. Hence, we get that:

$$\mathsf{Adv}^{\mathsf{code-bind}}_{\mathcal{B}_2,\mathsf{Com}}(\lambda) = \Pr[\mathsf{E}_{\mathsf{inv}}]$$

Lastly, we can construct $\mathcal{B}_3$ that breaks reconstruction binding in the event $\mathsf{E}_{\mathsf{inc}}$. This is the same as the one in Thm. 3.11 proof. Hence we get that:

$$\Pr[\mathsf{E}_{\mathsf{inc}}] \leq \mathsf{Adv}^{\mathsf{rec-bind}}_{\mathcal{B}_3,\mathsf{Com}}(\lambda) + \mathsf{Adv}^{\mathsf{extr}}_{\mathcal{B}_4,\mathsf{Com}}(\lambda).$$

Lastly, we analyze the probability of $\mathsf{E}_{\mathsf{ne,o}}$. Let us first bound the probability that all the samples of a particular subset of $\delta\ell$ verifiors fall into a particular subset of $1-\Delta+f$ fraction of symbols. We get,

$$(1-\Delta+f)^{\delta\ell Q},$$

since each query of each verifior is sampled uniformly at random from $[M]$. Next, we union bound over all possible subsets of $\delta\ell$ verifiors, and all possible subsets of $(1-\Delta+f)M$ symbols, to get:

$$\Pr[\mathsf{E}_{\mathsf{ne,o}}] \leq \binom{\ell}{\ell\delta} \cdot \binom{\mathsf{M}}{(1-\Delta+\mathsf{f})\mathsf{M}} \cdot (1-\Delta+\mathsf{f})^{\ell\delta\mathsf{Q}}$$

This proves the theorem. □

C Repair Safety Definition

Definition C.1. We say that a data availability sampling scheme DAS is (δ, f)-repair safe if for all PPT adversaries $\mathcal{A}$, the following probability is negligible in λ:

$$\mathsf{Adv}^{\mathsf{repair}}_{\mathcal{A},\mathsf{DAS},\delta,f}(\lambda) = \Pr[\mathbf{G}^{\mathsf{repair}}_{\mathcal{A},\mathsf{DAS},\delta,f}(\lambda) = 1]$$

where the probability is defined over the randomness of the adversary, the random coins of the Verify protocol, and the game is defined in Fig. 14.

Game $\mathbf{G}^{\text{repair}}_{\mathcal{A},\text{DAS},f,\delta}$ with respect to an adversary $\mathcal{A}$.

$\text{pp} \leftarrow\!\$\ \text{Setup}(1^\lambda)$
$(n, \text{com}, C, \{d_i\}_{i\in[n]\setminus C}, \{O_i(\cdot)\}_{i\in C}, \text{st}) \leftarrow\!\$\ \mathcal{A}(\lambda, \text{pp})$
$\forall\ j \in [\ell], (s_j, t_j) \leftarrow\!\$\ \text{Verify}(\text{com})\langle\{O_i(\text{st}, j, \cdot)\}_{i\in C}, \{\text{N}_i(d_i, \cdot)\}_{i\in[n]\setminus C}\rangle$
$s = \Sigma_{j\in[\ell]} s_j$ / added as integers
$\text{GoodSamp} := (s \geq \delta\ell)$
while $\exists\ i \in [n] \setminus C$ s.t. $d_i = \perp$: / Attempting to iteratively repair all chunks
  $r \leftarrow 0$
  $\forall\ i \in [n] \setminus C$ s.t. $d_i = \perp$:
    $d_i \leftarrow \text{Repair}(\text{com}, i)$
    if $\text{ChunkVerify}(\text{com}, i, d_i) = 1$ **then** $r \leftarrow 1$
  if $r = 0$ **then endwhile** / Repair didn't succeed for any node in this iteration
$\text{BadRepair} := (\exists\ i \in [n] \setminus C \text{ s.t. } d_i = \perp)$
return $\text{GoodSamp} \wedge |C| \leq fn \wedge \text{BadRepair}$

Fig. 14. The repair safety game $\mathbf{G}^{\text{repair}}_{\mathcal{A},\text{DAS},f,\delta}$ for a Data availability sampling scheme DAS. It is parametrized with respect to f, the fraction of corrupt nodes, and δ, denoting the fraction of verifiors that need to succeed in order for sampling to be successful. We use ChunkVerify to denote a deterministic algorithm which, given a commitment, and a chunk of a party, outputs 1 if the chunk is valid with respect to the commitment.