Committing AVID with Partial Retrieval and Optimal Storage

Nicolas Alhaddad
Boston University
USA
nhaddad@bu.edu

Leonid Reyzin
Boston University
USA
reyzin@bu.edu

Mayank Varia
Boston University
USA
varia@bu.edu

ABSTRACT

Asynchronous Verifiable Information Dispersal (AVID) allows a dealer to *disperse* a message M across a collection of server replicas consistently and efficiently, so that anyone can reliably *retrieve* the message M if some servers fail. AVID has direct applications to long-term file outsourcing, and is also useful as a building block in other distributed algorithms. Since AVID was introduced by Cachin and Tessaro in 2005, several works improved the asymptotic communication complexity of AVID protocols. However, recent gains in communication complexity have come at the expense of increased storage and the need to retrieve the entire message even if only part of it is sought.

This work contributes a new AVID construction that achieves optimal storage, support for retrieving only parts of the message, and guaranteed output delivery while maintaining optimal asymptotic communication complexity during dispersal and retrieval. This is accomplished by combining a few technical innovations. We obtain optimal storage by separating the honest thresholds needed for consensus and for retrieval. We support partial retrievals and guaranteed delivery by using bi-dimensional erasure coding and a novel committing technique that combines vector and polynomial commitments to detect malicious behavior by the dealer at dispersal time, whereas prior work deferred this detection to retrieval.

KEYWORDS

Distributed Systems, Consensus Protocols, Asynchronous Verifiable Information Dispersal, Optimal Storage, Partial Retrieval, AVID

1 INTRODUCTION

How can one distribute a file across a collection of outsourced servers such that anyone can retrieve the file at a later time, even if some of the servers have lost or corrupted data by then? Reliable methods for information dispersal are foundational to the design of distributed, fault-tolerant systems. Introduced by Rabin [23], verifiable information dispersal allows a dealing-client to *disperse* a file among a set of server replicas. It provides availability in the sense that the outsourced file can subsequently be *retrieved* by anyone if enough server replicas are honest, and integrity in the sense that everyone retrieves the same file even if some of the server replicas are faulty.

While this work focuses on the use of information dispersal for outsourced file storage, this primitive also lies at the heart of many applications involving distributed storage and fault-tolerant state-machine replication [12, 16]. For instance, verifiable information dispersal can be used to construct Byzantine reliable broadcast and agreement protocols [3, 12] in both synchronous and asynchronous networks, as well as privacy-preserving primitives like verifiable secret sharing [4, 26], distributed key generation [19], robust secure multi-party computation [21], and more.

AVID. More specifically, the focus of this work is *asynchronous verifiable information dispersal*, or AVID. This primitive was introduced by Cachin and Tessaro in [8], and their target scenario was outsourcing of files to storage area networks. They consider server-side storage devices ("replicas") whose goal is to store data in an efficient and fault-tolerant way. To this end, Cachin and Tessaro's definition of AVID explicitly allows for Byzantine failures of storage replicas (in this work, we additionally consider fail-stop replicas). Because their definition is meant to model storage devices that are decentralized and possibly geographically distributed, all communication is modeled as asynchronous.

Concretely, the goal of AVID is for a dealing-client to *disperse* a message M such that each server in a collection of n server replicas stores something shorter than M (called a "fragment"). At some later time, any client (not necessarily the original dealing-client) can *retrieve* a collection of fragments from k honest replicas and up to t malicious replicas, and use this data to reconstruct M. The requirements of an AVID protocol combine the correctness requirement of an error-correcting code (e.g., that k fragments suffice to reconstruct a message) with several consensus requirements (e.g., the replicas will agree on whether dispersal has been completed and clients will agree on the retrieved message). The requirements also protect against the possibility that the dealing-client is *malicious*, meaning that they disperse fragments that are not consistent with any message.

Prior constructions. Cachin and Tessaro's AVID construction meets their security definitions, but in an inefficient way: when dispersing a message, each of the n replicas eventually learns the entire message M in order to verify consistency of all fragments, before discarding most of it and only storing a smaller fragment of the message for later retrieval. As a result, their protocol has total communication complexity of at least $n \cdot |M|$. On the other hand, its storage complexity is optimal, at $|M|/k$ per replica.

Subsequently, a series of recent works [1, 2, 16, 25] improved the communication complexity of AVID using a common high-level design: encode the data into multiple fragments using an erasure-correcting code; send a fragment to each replica; reliably broadcast a commitment to the set of all fragments; and have each replica verify that its own fragment is consistent with the commitment. This design ensures that each replica receives data proportional to the size of its own fragment, so the overall communication complexity is $O(|M|)$ (plus the commitment overhead). However, all recent works suffer from four common flaws:

(1) there is no modeling of replicas that go off-line or lose their fragments between dispersal and retrieval—instead, a failed replica is treated as malicious, and it is subject to the upper bound of t total faults;

(2) the per-replica storage is worse than the optimal $|M|/k$ of Cachin-Tessaro [8];
(3) a retrieving-client must retrieve the entire message in order to perform the consistency check, even if they only wanted part of it; and
(4) the server replicas cannot detect if the dealing-client has provided fragments that are not consistent with any message (only a retrieving-client will learn this).

In this work, *we provide new definitions and constructions of AVID that overcome all of these limitations.*

1.1 Our Contributions

This work contributes four types of enhancements and clarifications to the definition of AVID, along with new AVID constructions that are optimal along all of these dimensions (see Table 1 for details) and desirable in the application of AVID to outsourced file storage.

AVID with fail-stop adversaries. The traditional definition of AVID assumes that all $n-t$ honest server replicas from the dispersal stage will also participate in retrieval, i.e., that $k = n - t$. However, this assumption may not be realistic in the setting of outsourced file storage, where retrieval can occur months or years later, during which time some server replicas might go offline or their storage devices can fail. To allow for this possibility, this work explicitly introduces d *fail stop-adversaries* into our network model in §3. As shown in Figure 1, these d replicas participate honestly in the dispersal phase, but then they are disconnected from the network so that only $k = n - t - d$ honest replicas and t malicious replicas are available to participate in retrieval.

We contribute an AVID construction that only requires an honest majority ($k > t$) during retrieval; by contrast, all AVID protocols (including ours) require a 2/3 supermajority of honest server replicas during dispersal. The high-level idea of our construction is as follows. Using an erasure code, the dealing-client encodes its message M into a collection of n fragments, and then encodes each fragment into a collection of n sub-fragments. The eventual goal of dispersal is for each replica i to learn fragment i. But at the start, the dealing-client instead disperses to each replica i the i^{th} sub-fragment of everyone's fragment (along with some commitments and proofs). Using the honest supermajority during dispersal, all server replicas learn their own fragment in just 3 rounds of communication. At retrieval time, an honest majority suffices to distinguish the commitment to M from the honest vs. malicious server replicas, and the erasure code can reproduce M from the k available fragments.

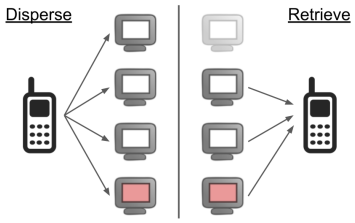

Figure 1: An AVID construction has two parts: a dealing-client disperses a message, and later another retrieving-client retrieves it. Among the n replicas, up to t replicas can be malicious and another d can fail in between disperse and retrieve. The remaining $k = n - t - d$ replicas honestly participate in retrieve. In the figure, $n = 4$, $k = 2$, and $t = d = 1$.

AVID with optimal storage. Next, in §4 we extend the above AVID construction to achieve the optimal storage size of $\frac{|M|}{k}$ per fragment (plus additive factors depending only on n and the security parameter λ). In this way, the parameter k allows for a tradeoff between support for accidental failures versus the size of each fragment; our fragments can be as small as size $\frac{|M|}{n-t}$ if no fail-stops are permitted, or up to size $\frac{|M|}{t+1}$ with the maximum tolerance for fail-stop adversaries.

We emphasize that support for optimal storage is a challenge to achieve even without fail-stops. Indeed, the recent prior works described above achieve the worst combination of $\frac{|M|}{t+1}$ storage cost and no fail-stop tolerance, as shown in Table 1. In §4.1, we explain why the server replicas must hold data of size $\frac{|M|}{t+1}$ at *some* point during dispersal, and therefore why the protocol design of all recent AVID constructions inherently cannot provide optimal storage.

To achieve optimal storage, our construction initially disperses a collection of sub-fragments of larger size $\frac{|M|}{t+1}$, but then constructs the fragments of smaller and optimal size $\frac{|M|}{k}$ by the end of dispersal. Our protocol also achieves optimal communication rounds, communication complexity (up to log factors), and only uses hash functions without the need for any trusted setup. The net result is a significant reduction in costs in the application of AVID to outsourced file storage, since the memory required during dispersal will often only be needed for just a few seconds or minutes (and anyway is no worse than prior constructions), and the long-term cost saving of storing small fragments on the server's hard drive accrues for many months to years afterward.

AVID with partial retrieval. Next, we define and construct AVID that allows for partial retrieval. Concretely in the setting of outsourced file storage, a dealing-client might disperse a large message M (e.g., a backup of an entire filesystem), but later might only need to retrieve a few *sub-blocks* of M at a time (e.g., just one file or directory).

Partial retrieval introduce two challenges: efficiency and consistency. Efficiency is straightforward: the network communication during retrieval must be proportional only to the size of the partial retrieval rather than the entire M. Our construction supports the ability for an retrieving-client to retrieve just some of the sub-fragments of M.

The much harder and more subtle issue with partial retrieval has to do with *retrieval consistency*. AVID already requires consistency across replicas: multiple retrieving-clients must receive the same (full) message, even if the adversarial server replicas act differently in each attempt at retrieval. But now we also require consistency across fragments that were dispersed by the (potentially dishonest) dealing-client. Concretely, if two clients retrieve different portions of the message, then we require that the retrievals must be consistent for all sub-blocks they have in common.

This requirement is incompatible with most recent AVID constructions, which allow a retrieving-client to output an "error" $\perp$ if it detects an inconsistency in *any* part of the retrieved message.

Table 1: Comparison of asynchronous verifiable information dispersal (AVID) protocols for a message M of length $|M|$ when the reconstruction threshold $k = n - t$. Cells in blue denote asymptotically optimal results, possibly up to polylog or additive factors in the security parameter λ or number of replicas n. The following acronyms are used in the table; DL: Discrete Logarithm, CRS: Common Reference String, q-SDH: q-Strong Diffie-Hellman, ROM: Random Oracle Model.

Scheme	Robust	Dispersal Cost (total)	Retrieval Cost (total)	Storage Cost (total)	Cryptographic Assumption	Setup
Cachin-Tessaro	yes	$O(n\|M\| + \lambda n^2 \log n)$	$O(\|M\| + \lambda n \log n)$	$\frac{n}{n-t}\|M\| + \lambda n \log n$	Hash	None
Hendricks et al.	no	$O(\|M\| + \lambda n^3)$	$O(\|M\| + \lambda n^2)$	$\frac{n}{t+1}\|M\| + \lambda n^2$	Hash + ROM	None
Alhaddad et al.	no	$O(\|M\| + \lambda n^2)$	$O(\|M\| + \lambda n \log n)$	$\frac{2n}{t+1}\|M\| + \lambda n \log n$	DL + ROM + Hash	CRS
Alhaddad et al.	no	$O(\|M\| + \lambda n^2)$	$O(\|M\| + \lambda n)$	$\frac{2n}{t+1}\|M\| + \lambda n$	q-SDH+Hash	Trusted
DispersedLedger	no	$O(\|M\| + \lambda n^2)$	$O(\|M\| + \lambda n \log n)$	$\frac{n}{t+1}\|M\| + \lambda n \log n$	Hash	None
Alhaddad et al.	no	$O(\|M\| + \lambda n^2)$	$O(\|M\| + \lambda n)$	$\frac{n}{t+1}\|M\| + \lambda n$	Hash	None
This work (§4)	no	$O(\|M\| + \lambda n^2 \log n)$	$O(\|M\| + \lambda n \log n)$	$\frac{n}{n-t}\|M\| + \lambda n \log n$	Hash	None
This work (§5 + Bulletproofs)	yes	$O(\|M\| + \lambda n^2 \log n)$	$O(\|M\| + \lambda n \log n)$	$\frac{n}{n-t}\|M\| + \lambda n \log n$	DL + ROM + Hash	CRS
This work (§5 + KZG)	yes	$O(\|M\| + \lambda n^2)$	$O(\|M\| + \lambda n)$	$\frac{n}{n-t}\|M\| + 2\lambda n$	q-SDH + Hash	Trusted

Hence, a client who partially retrieves only one sub-block must somehow return $\perp$ even if there is an inconsistency in another sub-block, which is inefficient. To resolve this issue, we introduce the idea of robust AVID.

Robust and Committing AVID. Our robustness guarantee to AVID states that successful completion of dispersal ensures that the fragments are consistent with the encoding of some message that can be subsequently retrieved. This requirement is similar to the robustness/guaranteed output delivery property of secure multi-party computation [21, 26], which we formalize in Def. 2.

We contribute a robust AVID construction using a novel encoding scheme based on bi-dimensional erasure coding. Concretely, we construct a variant of Reed-Solomon codes where the codeword corresponding to message M is a bivariate polynomial $P(\cdot, \cdot)$ of degree $n - 2t - 1$ in one dimension and $k - 1$ in the other. In this case, a "sub-fragment" is one point $P(i, j)$ on the curve, and a "fragment" is a univariate polynomial $P(\cdot, j)$ formed by interpolating all points received from other replicas. We show that this structure results in *bi-dimensional* Reed-Solomon erasure coding over bivariate polynomials (as shown in Figure 3), which may be of independent interest.

Lastly, we introduce the concept of committing AVID based on the following observation: if the retrieving-client happens to know the polynomial commitment to P, then we can support a *dishonest majority* of $t > k$ during retrieval time. This assumption is already common in outsourced file scenarios, where either the dispersing- and retrieving-client are the same entity (e.g., in file backup), or the two clients have a separate authenticated but low-bandwidth communication channel (e.g., retrieving a checksum from a website before torrenting a large file). The commitment provides a natural "handle" or pointer that the retrieving-client can use to request the correct file. In this way, robust and committing AVID is similar to the "provable retrievability" property in the recent work of Nazirkhanova, Neu, and Tse [22], with the important difference that they rely on the assistance of a blockchain to disperse the commitment and achieve consensus between servers, whereas we merely require that there exists some mechanism for the retrieving-client to retrieve one succinct commitment.

1.2 Related Work

The concept of verifiable information dispersal was introduced by Rabin in 1990 [23], and its asynchronous counterpart was introduced by Cachin and Tessaro in 2005 [8]. We provide a direct comparison between our work and prior AVID constructions in Table 1. Beyond its value as a standalone primitive, AVID has applications to the design of distributed algorithms, blockchains and state-machine replication, secure computation, and outsourced file storage.

Distributed algorithms. Recent works have shown a two-way relationship between AVID and other consensus-building protocols like Byzantine reliable broadcast [6]: AVID and its generalizations can be used in a black-box manner to construct Byzantine reliable broadcast (e.g., [3, 12]), and most AVID constructions (including ours) use reliable broadcast as an internal building block to reach agreement on a short commitment to the message. AVID is also used as a fundamental building block within asynchronous, Byzantine fault-tolerant protocols like DispersedLedger [25]. Our construction can be a drop-in replacement that withstands fail-stop adversaries, reduces storage cost, and provides robustness (see Table 1).

Blockchains and state-machine replication. Recent work by Nazirkhanova, Neu, and Tse [22] introduces a robustness-style guarantee for AVID based on a blockchain: each replica posts an attestation that its fragment is consistent with the commitment C, and if enough attestations are posted then the message is known to be retrievable. Our work provides the same guarantees and also reduces on-chain storage by a factor of two. Our AVID construction can be a drop-in replacement in their application to rollups.

Secure computation. Several works have used AVID and reliable broadcast protocols in order to construct privacy-preserving protocols. Both Cachin-Tessaro [8] and Alhaddad et al. [4] show how to construct asynchronous verifiable secret sharing protocols through the use of AVID-dispersal of an encrypted message combined with secret-sharing of the corresponding cryptographic key. Additionally, several works show how AVID and asynchronous verifiable secret sharing can be combined in order to construct asynchronous, robust protocols for key agreement or general-purpose secure multi-party computation [13, 19, 21].

Outsourced file storage. Related to our robustness notion, many works study the problem of verification that outsourced servers are storing uploaded files, for both single-server (e.g., [17]) and multi-server (e.g., [10, 11, 14, 15]) cases. AVID offers an incomparable guarantee: while it relies on the assumption that there is an upper bound on the number of corrupted servers, it allows the dealing-client and retrieving-client to be different, because no client needs to maintain any local storage. Moreover, AVID offers protection against a malicious dealing-client (and robust AVID offers even stronger protection).

2 BUILDING BLOCKS

Crypto primitives. This work uses three primitives from cryptography that we briefly describe. First, we use a hash function hash, which we model as a random oracle [5].

Second, we use a Merkle tree as an instantiation of a deterministic vector commitment scheme, which creates a short digest c that binds a list of data $\vec{v}$ and allows the committer to prove list membership. Concretely, a vector commitment scheme has four methods: setup of public parameters $\overline{\text{pp}}$, producing a vector commitment $\text{VCom}(\overline{\text{pp}}, \vec{v}) \to c$, producing a proof $\text{VGen}(\overline{\text{pp}}, \vec{v}, i) \to \pi_i$ that v_i is in $\vec{v}$ at location i, and verifying this proof of list membership via $\text{VVerify}(\overline{\text{pp}}, c, v_i, \pi) \to$ True/False (cf. [9, 20] for details).

Third, we use polynomial commitment schemes to create a digest of a polynomial $\text{Com}(\text{pp}, \phi(x), d) \to \hat{\phi}$ in such a way that the committer can generate a proof $\text{open}(\text{pp}, \phi, i) \to \langle i, \phi(i), \pi_i \rangle$ that a point lies on the polynomial, which can be publicly verified via $\text{Verify}(\text{pp}, \hat{\phi}, y, d)$. We use two instantiations of polynomial commitment schemes in this work: bulletproofs with constant-size commitments and log-size proofs [7], and KZG commitments with constant-size proofs based on trusted setup [18].

In this section, we introduce the primitives that are required for this construction: hash functions, vector and polynomial commitments, and erasure codes.

Hash Function. A cryptographic collision-resistant hash function H guarantees that a computationally bounded adversary cannot come up with two inputs that hash to the same value, except with negligible probability. In this work, we model a hash function as a random oracle [5].

Following the convention in distributed algorithms papers, in our security analysis we will actually assume that the probability of breaking this or any other cryptographic building block is zero; that said, it is straightforward to extend our analyses to a more traditional reduction to the security of its fundamental primitives.

Vector Commitments. A deterministic *vector commitment scheme* $\mathcal{V} = (\text{VSetup}, \text{VCom}, \text{VGen}, \text{VVerify})$ comprises four algorithms that operate as follows:

- $\text{VSetup}(1^\lambda, U, n) \to \overline{\text{pp}}$ is given a security parameter λ, a set U, and a maximum vector length n. It generates public parameters $\overline{\text{pp}}$.
- $\text{VCom}(\overline{\text{pp}}, \vec{v}) \to c$ is given a vector $\vec{v} \in U^\ell$ where $\ell \le n$. It outputs a commitment string c.
- $\text{VGen}(\overline{\text{pp}}, \vec{v}, i) \to \pi_i$ is given a vector $\vec{v}$ and an index i. It outputs a proof string π_i.
- $\text{VVerify}(\overline{\text{pp}}, c, u_i, \pi) \to$ True/False takes as input a vector commitment c, an indexed element $u_i \in U$, and a proof string π. It outputs True if $u_i = \vec{v}[i]$ and π is a witness to this fact and False otherwise.

A well-known example of a vector commitment is a Merkle tree, and several other constructions exist (e.g., [9, 20]).

Polynomial Commitments. A *polynomial commitment scheme* $\mathcal{P}$ comprises several algorithms described below, the last three of which are optional.

- $\text{Setup}(1^\lambda, \mathbb{F}, D) \to \text{pp}$ is given a security parameter λ, a finite field $\mathbb{F}$, and an upper bound D on the degree of any polynomial to be committed. It generates public parameters pp that are required for all subsequent operations.
- $\text{Com}(\text{pp}, \phi(x), d) \to \hat{\phi}$ is given a polynomial $\phi(x) \in \mathbb{F}[x]$ of degree $d \le D$. It outputs a commitment string $\hat{\phi}$ (throughout this work, we use the hat notation to denote a commitment to a polynomial).
- $\text{open}(\text{pp}, \phi, i) \to \langle i, \phi(i), \pi_i \rangle$ is given a polynomial ϕ as well as an index $i \in \mathbb{F}$. It outputs a 3-tuple containing i, the evaluation $\phi(i)$, and a proof π_i that attests that $\phi(i)$ is an evaluation of the polynomial ϕ at i .
- $\text{Verify}(\text{pp}, \hat{\phi}, y, d) \to$ True/False takes as input a commitment $\hat{\phi}$, a 3-tuple $y = \langle i, j, w \rangle$, and a degree d. It outputs a Boolean.
- $\textit{open}_{batch}(\text{pp}, [\phi_1 \dots \phi_n], i) \to \langle i, [\phi_1(i) \dots \phi_n(i)], \pi_i \rangle$ is given a list of n polynomials $[\phi_1 \dots \phi_n, i]$ as well as an index $i \in \mathbb{F}$. It outputs a 3-tuple containing i, the n evaluations $[\phi_1(i), \dots \phi_n(i)]$, and the batched proof π_i.
- $\textit{Verify}_{batch}(\text{pp}, [\hat{\phi}_1 \dots \hat{\phi}_n], \langle i, [\phi_1(i) \dots \phi_n(i)], \pi_i \rangle, d) \to$ True/False takes as input a list of n polynomial commitments $[\hat{\phi}_1 \dots \hat{\phi}_n]$, a 3-tuple $y = \langle i, [\phi_1(i) \dots \phi_n(i)], \pi_i \rangle$, consisting of an index, n evaluations at i (each at a different polynomial), a batch proof π_i and a degree d. It outputs True if the evaluations are on the polynomials and False otherwise.
- $\textit{open}_{agg}(\text{pp}, \phi, [x_1, \dots x_n]) \to \langle [x_1, \dots x_n], [\phi(x_1) \dots \phi(x_n)], \pi \rangle$ is given a list of indices $[x_1, \dots x_n]$ where every index $x_i \in \mathbb{F}$ and a polynomial ϕ. It outputs a 3-tuple containing the indices $[x_1, \dots x_n]$, the n evaluations $[\phi(x_1), \dots \phi(x_n)]$, and the aggregate proof π.
- $\textit{Verify}_{agg}(\text{pp}, \hat{\phi}, \langle [x_1, \dots x_n], [\phi(x_1) \dots \phi(x_n)], \pi \rangle) \to$ True/False takes as input a polynomial commitment $\hat{\phi}$ and a 3-tuple $y = \langle [x_1, \dots x_n], [\phi_1(i) \dots \phi_n(i)], \pi_i \rangle$, consisting of list of n indices $[x_1, \dots x_n]$ where every index $x_i \in \mathbb{F}$, n evaluations at each index x_i, an aggregate proof π. It outputs True if the evaluations are on the polynomial at the right indices and False otherwise.
- $\text{Hom}(\text{pp}, \hat{\phi}_1, \hat{\phi}_2, a) \to \widehat{\phi_1 + a\phi_2}$ takes in commitments to two polynomials ϕ_1 and ϕ_2 of degree at most D, as well as a field element $a \in \mathbb{F}$. It outputs the commitment $\text{Com}(\text{pp}, \phi, \max\{d_1, d_2\})$ corresponding to the polynomial $\phi = \phi_1 + a\phi_2$.

In our constructions, we abuse notation and omit the use of Setup, VSetup (assuming they have already been performed beforehand) and public parameters $\overline{\text{pp}}$ and pp (considering them implicitly given to each algorithm). Also: throughout this work, we do not require

that any commitments are hiding. Concretely, we consider two instantiations of polynomial commitments throughout this work: KZG commitments with constant-size proofs based on trusted setup [18], and bulletproofs with log-sized proofs [7].

Erasure coding. An erasure code transforms a message into a larger *codeword* that contains a total of *n fragments*, such that the message is retrievable even if only k fragments remain available. Throughout this work, we consider erasure codes that are *systematic*, meaning that the concatenation of the first k coded fragments equals the original message.

Formally, a (k, n)-systematic erasure coding scheme consists of two algorithms, *encode* and *decode*, that are defined with respect to an alphabet $\mathcal{F}$. The encoding algorithm $encode(M, k, n)$ takes a message M and outputs n fragments $(f_1, \ldots f_n)$, where each fragment is in $\mathcal{F}$ and such that any k of those fragments can be used to reconstruct M via the decoding algorithm. The decoding algorithm takes a list of k fragments and their indices $(i_1, f_{i_1}), (i_2, f_{i_2}), \ldots (i_k, f_{i_k})$ (where all i_j values are distinct) and returns M. To emphasize the number of fragments needed to decode, we will use notation $decode(k)$. Concretely, in this work we will use erasure codes based on Reed-Solomon [24]. Its alphabet is a finite field $\mathcal{F} = \mathrm{GF}(q)$, where q is a prime power and $q \geq n$.

3 SYSTEM MODEL AND DEFINITIONS

This section formally defines our threat model with malicious and fail-stop replicas, formally defines AVID, describe our metric for storage blowup, and adds a new robustness guarantee on top of AVID.

3.1 Network Assumptions and Threat Model

We consider a set of n server *replicas* connected by a network, such that there exists an authenticated channel between each pair of (honest) replicas, and also between replicas and external participants called *clients*. They communicate over an asynchronous network as follows: a dealing-client only sends one message to each server replica, a retrieving-client only sends and receives one message with each server replica, and the server replicas can interact with each other as many times as they wish.

In more detail, we define a (t, d)-*limited adversary* as an actor with three types of powers. First, the adversary can reorder and arbitrarily delay communications transmitted through the asynchronous network, but it cannot drop packets (i.e., messages sent between honest replicas will eventually be delivered, but there is no known upper bound on how long this may take). Second, as with prior works in this space, we also assume that the adversary can adaptively corrupt up to t replicas during the *avid-disperse* and *avid-retrieve* protocols; corrupted replicas can deviate from the protocol arbitrarily, sending any message of the adversary's choice or refusing to send a message at all. A corrupted replica is called *malicious*; all other replicas are called *honest*. Third, unlike prior works in this space, we additionally give the adversary the power to disconnect up to d additional replicas during each invocation of the *avid-retrieve* protocol. In other words, the adversary can induce d so-called "fail-stop" failures during *avid-retrieve*. These replicas are *disconnected* (but not malicious); the remaining ones are *available*.

Disconnected replicas will not participate in the instance of *avid-retrieve*, and therefore the retrieving client will never hear from them. Thus, while at least $n - t$ replicas participate honestly in *avid-disperse*, the retrieving client can expect to hear from at most $k = n - t - d$ honest replicas along with the t Byzantine corrupted ones. Note that there may be multiple instances of *avid-retrieve* executed after a single *avid-disperse* (e.g., to different clients), and, in our model, the adversary can disconnect a different subset of d replicas for each instance of *avid-retrieve*.

Fail-stop adversaries can model long-term failure or attrition of server replicas that occur in the (possibly lengthy) time interval between *avid-disperse* and *avid-retrieve*. They also model extreme delays of the asynchronous network that prevent some of the d server replicas from completing *avid-disperse* before a retrieving-client starts *avid-retrieve*.

We note that the parameter k of participants in *avid-retrieve* appears in the original AVID definition by Cachin and Tessaro [8], although they do not explicitly address fail-stop corruptions. The parameter k is not present in many subsequent works (see, e.g., [25]); adding this parameter back into the definition allows us to have a more careful accounting of storage costs, as we discuss below.

3.2 Defining AVID with Partial Retrieval

For any (t, d)-limited adversary, an asynchronous verifiable information dispersal (AVID) scheme consists of a pair of protocols *avid-disperse* and *avid-retrieve*. The dispersal protocol is initiated by a client (called a *dealing-client*) with a message M; it disperses M among n replicas. We will denote this protocol $avid\text{-}disperse(M)$. For syntactic convenience, when replica i finishes the dispersal protocol, we will say that it calls a special instruction $avid\text{-}deliver(m_i)$, where m_i is the information that replica i saves in its long-term storage, and subsequently uses as its input in *avid-retrieve*.

The retrieval protocol *avid-retrieve* allows any client to fetch a portion of the message M. In more detail, we let $M = [M_1, M_2, \ldots, M_\alpha]$ be the concatenation of α *sub-blocks*, and we allow the client to retrieve M_x at any index x. This client is called a *retrieving-client*, and it communicates with a subset of replicas holding M, with k honest and up to t malicious replicas. The retrieval protocol initiates when a client calls $avid\text{-}retrieve(x)$ for an index x, and completes when the client reconstructs the message sub-block M_x; upon completion, we say that the retrieving-client calls a special instruction $avid\text{-}output(M_x)$. We emphasize the following: (1) the retrieving-client need not be the same entity as the dealing-client, (2) multiple retrieving-clients can retrieve different sub-blocks of M after a single dispersal, and (3) while our definition only considers retrieval of a single sub-block for simplicity, it is straightforward also to consider efficient retrieval of many or all sub-blocks at once.

Throughout this work, we will assume that *avid-disperse* and *avid-retrieve* are deterministic algorithms. Additionally, we model all replicas as interactive Turing machines that connect to each other and optionally also have access to a structured common reference string. We do not consider any per-replica initialization such as a PKI within this work.

Definition 1. An (n, t, k) AVID protocol for a message space $\mathcal{M} = B^\alpha$ (where $B = \{0, 1\}^\beta$ is the collection of sub-block strings) satisfies

the following properties with overwhelming probability for every $M \in \mathcal{M}$ against a (t, d)-limited adversary, where $d = n - t - k$:

- **Termination**: If an honest dealing-client initiates *avid-disperse* on a message M, then *avid-deliver* is eventually completed by all honest replicas.
- **Dispersal Agreement**: If an honest replica completes *avid-deliver*, then all honest replicas eventually complete *avid-deliver*. (This property holds whether the dealing-client is honest or malicious.)
- **Availability**: Any honest retrieving-client who initiates *avid-retrieve*(x) eventually reconstructs some message sub-block M_x, as long as k replicas who have completed *avid-deliver* remain honest and available throughout *avid-retrieve*.
- **Correctness**: If an honest dealing-client initiates *avid-disperse*(M) with $M = [M_j]_{j \in \{1,2,\ldots,\alpha\}}$, then an honest retrieving-client who initiates *avid-retrieve*(x) for any index x eventually completes *avid-output*(M_x).
- **Retrieval Consistency**: If at least one honest replica completes *avid-deliver*, then all honest retrieving-clients who initiate *avid-retrieve*(x) at any index x will eventually call *avid-output*(M_x) for the same M_x. (This property holds whether the dealing-client is honest or malicious, and regardless of which d replicas are disconnected in between *avid-disperse* and *avid-retrieve*(x).)

We note that in prior work, correctness and retrieval consistency are often written together as a single property (e.g., [8]). We separate them in order to distinguish more clearly between the cases of an honest and malicious dealing-client.

We remark that this definition is deceptively powerful; indeed, the writing in Def. 1 nearly matches the original definition of Cachin-Tessaro [8], and it is instead the change in the adversary model that gives these properties new meaning. We highlight several implications of Definition 1 that go beyond prior definitions of AVID. First, note that the concept of availability stands alone and does not depend on the definition of agreement; in particular, *avid-retrieve* may start even before all honest replicas complete *avid-deliver*. Second, retrieval consistency must hold even in the presence of fail-stop adversaries, so information theoretically any k replicas must hold $|M|$ bits of data between them after dispersal. Third, fail-stop adversaries and optimal storage together require moving away from the structure of previous AVID constructions (see §4.1 for details) and necessitates a more "balanced" construction where each honest replica completes dispersal with a $\frac{|M|}{k}$-sized fragment of the data. Fourth, this definition can only be achieved when $k \in [t + 1, n - t]$ due to the lack of a PKI; the upper bound follows directly from the fact that k is a count of honest replicas, and the lower bound is shown below.

Theorem 1. *For any n, if $k \leq t$, then there does not exist an (n, t, k) AVID construction that meets Definition 1.*

The key idea here is that because the retrieving client did not participate in dispersal and may not even know who the dispersing client is, so it is relying on a majority vote to distinguish fragments from honest vs. malicious replicas. See Appendix B for a full proof.

3.3 Storage Complexity

We will adopt the same definitions and model for measuring storage complexity as with the original work of Cachin and Tessaro [8]. We will re-formulate them in the following paragraph for clarity. We distinguish between two types of memory: 1) short term working memory that is used during *avid-disperse* but freed when *avid-disperse* finishes, and 2) long term storage that the replicas must maintain after the completion of *avid-disperse*, to enable useful participation in *avid-retrieve*. We make this distinction because working memory can be reused once *avid-disperse* is finished, whereas long term storage needs to be occupied for as long as the data may be needed. In this work, as in Cachin-Tessaro [8], we focus on long term storage as our metric of storage complexity. Formally:

- The **storage complexity** for replica i is the bit length of m_i when *avid-deliver*(m_i) is called. The overall storage complexity of an information dispersal scheme is the maximum (over the choice of $M \in \mathcal{M}$ and adversarial behavior) of the sum of storage storage complexities of all honest replicas.
- The **storage blow-up** is the ratio of the storage complexity and $|M|$, in the limit as $|M| \to \infty$.

As mentioned in Cachin-Tessaro [8], the smallest possible storage blow-up of an (n, t, k) AVID is $\frac{n}{k}$, because any set of k replicas needs to store the entire message M among them, and there are n replicas total.

3.4 Defining Robust and Committing AVID

The retrieval consistency property of AVID guarantees that the retrieved message will be the same regardless of which d replicas fail to participate in the retrieval and how t malicious replicas behave, even if the dealing-client is malicious. This property is designed to ensure that a malicious dealing-client cannot cause different output messages. However, it does not say that the output message is correct—the corectness property only applies to an honest dealing-client, and the meaning of "correct retrieval" is unclear when the dealing-client is malicious, because the correct message may not even be well-defined. In fact, the m_i values in *avid-deliver*(m_i) may not even be consistent with any M.

Some past work has dealt with this issue by having *avid-retrieve* output a special character (like $\perp$) when inconsistency is detected (e.g., [1, 2]). This ensured consistency for full message retrieval, but this strategy is not conducive to partial retrieval. We take a different approach and insist that an inconsistent dealing-client is rejected early, at the *avid-disperse* stage, rather than during *avid-retrieve*. This sidesteps the need for any fallback error messages. Formalizing this property is somewhat nontrivial: after all, a malicious dealing-client has limitless options, some of which appear very close to honest.

In Def. 2, we provide a formalization that is well-defined as long as *avid-disperse* is deterministic. It says that if the honest replicas complete *avid-disperse* and subsequently use these fragments in *avid-retrieve* to reconstruct some message M, then their fragments must all be what an honest dealing-client *would have produced* when invoked on message M.

Definition 2. A (n, t, k)-AVID protocol is *robust* against a (t, d)-limited adversary A if retrieval consistency is strengthened as follows. If an honest retrieving-client initiates *avid-retrieve* on all indices x to recover some message $M = [M_x]$, and then subsequently acts as an honest dealing-client and runs *avid-disperse*(M), then each replica i must complete it with *avid-deliver*(m_i)—that is, with the same output m_i as in the initial dispersal of M—regardless of the behavior of malicious replicas.

Next, we extend this definition to give the adversary the power to *corrupt* an additional d replicas during the *avid-retrieve* protocol. In other words, we now consider the d replicas to be Byzantine failures rather than fail-stop failures, and therefore it is possible that a *dishonest majority* of server replicas participate in retrieval (i.e., $t + d > k$).

This begs the question: can an AVID protocol exist if the reconstruction threshold k is less than t? Here there is good news and bad news. Unfortunately, we have showed in Theorem 1 that it's not possible to support a dishonest majority with the traditional AVID definition (whether robust or not). The good news is that a minor change to the definition suffices to support retrieval with dishonest majority.

The fundamental challenge is that without an honest majority in retrieval, there cannot be consensus over what message has been dispersed. Therefore, a natural course of action is to relax the requirement for *avid-retrieve*, so that no consensus has to be solved. We assume that the retrieving-client has *a priori* received one small piece of information from the dealing-client— namely, a succinct binding commitment to the message being dispersed. This way, even if more malicious replicas show up to the retrieval phase, they would not be able to convince the fetching client of a different message as it would require the malicious replicas to break the binding property of the commitment.

Hence, we introduce a new definition for AVID called *robust committing AVID*. The new definition makes changes to both dispersal and the retrieval. In the dispersal, we require every replica to deliver a succinct binding commitment c alongside its fragment. In the retrieval phase, the client now starts with c as an input; we presume that this commitment has already been transmitted accurately out-of-band. We present our formal definition below: To achieve such a strong guarantee, we must assume that the retrieving-client has *a priori* received one small piece of information from the dealing-client: a succinct, binding commitment c to the message being dispersed. We note that the out-of-band exchange of c is a plausible requirement to impose. For instance, in the outsourced file storage setting, the fetching client must have some identifier to reference the message that it wants to retrieve, and one can think of the succinct commitment as that identifier.

Definition 3. A (n, t, k)-robust AVID protocol is *committing* if retrieval consistency is changed to the following statement: if at least one honest replica completes *avid-deliver* with commitment c, then all honest retrieving-clients who initiate *avid-retrieve*(x, c) at every index x will eventually retrieve the same message $M = [M_x]$, and $Com(M) = c$. (This property holds whether the dealing-client is honest or malicious, and regardless of which d replicas are disconnected in between *avid-disperse* and *avid-retrieve*(x).)

Moreover, having a succinct commitment allows for AVID to support additional use cases. Consider the realm of blockchains, in which blocks can be dispersed among different nodes and stored for the long term, which cuts down on storage costs. Specific blocks could be retrieved from the storage nodes, with the retrieving client having the ability to verify the correctness of any block, even though an honest majority might not be present. As a result, a committing AVID can support the application to rollups described by Nazirkhanova, Neu, and Tse [22].

4 CONSTRUCTING AVID WITH OPTIMAL STORAGE

In this section, we construct an AVID protocol with optimal storage against fail-stop adversaries. While the construction does use a commitment scheme, it does not guarantee that a retrieving-client receives a message corresponding to this commitment (i.e., it allows for a failure output $\perp$).

4.1 The Challenge of Achieving Optimal Storage

To motivate our construction, we first explain why having the optimal storage blowup of $\frac{n}{k}$ is difficult to achieve, even for a fixed $k = n - t$ (i.e., without additional fail-stops).

Recent AVID protocols with $O(|M|)$ communication complexity for $t \leq \frac{n-1}{3}$ all have **storage blow-ups** of at least $\frac{n}{n-2t}$ even though $k = n - t$; this not by accident, and in fact it is required for their constructions. In all recent work the dispersal phase has the same basic structure:

(1) Commit to the data with a commitment,
(2) Reliably broadcast the commitment, and
(3) Send erasure coded fragments of the data to all replicas and show how they link to the commitment.

Replicas agree among themselves whether the erasure coded fragments they received are consistent with the commitment. To this goal, sometimes Steps 2 and 3 are intertwined so that all replicas finish the broadcast once **enough** erasure coded fragments are received.

If a protocol follows the above structure using an erasure code whose fragments are of size $\frac{|M|}{n-t}$ (if $n = 3t + 1$, which is known to be most failures for which consensus is still achievable, that would roughly double their storage efficiency relative to the size $\frac{|M|}{t+1}$ that they actually use), then we claim that the protocols would run into trouble. The issue is that a malicious dispersing client can send valid shares to $n - 2t$ honest replicas, but nothing at all to the remaining t honest-but-clueless replicas. The t malicious replicas can also behave correctly in all interactions with the honest replicas, causing them to complete the dispersal phase (since from their perspective, the t clueless replicas appear to be the malicious ones). Information theoretically, the honest replicas have $\frac{n-2t}{n-t}|M|$ bits of data and therefore cannot reconstruct M on their own; additionally, the clueless replicas never complete dispersal and cannot help them. This leaves the honest replicas at the mercy of the malicious replicas to provide the remaining data in ways that could lead to retrieval of multiple possible messages, which breaks retrieval consistency in Def. 1.

In order to have optimal storage blow up, each replica must store $O(\frac{|M|}{n-t})$ bits. However, as we have seen, naively letting the dealing-client send each replica $\frac{|M|}{n-t}$ size fragments would not work in an asynchronous system. One way around this problem is to reliably broadcast the whole message and then let each replica save its own $\frac{|M|}{n-t}$ share of the message. This approach was taken by the work of Cachin-Tessaro [8], but it leads to a communication complexity of at least $O(n \cdot |M|)$ because each replica has to have the full message.

In summary, any AVID protocol that achieves a communication complexity of $O(|M|)$ and per-node storage of $\frac{|M|}{n-t}$ must have all of the following properties.

- Since dispersal can complete if only $n - 2t$ honest replicas receive valid data from a malicious dealing-client, these replicas must send enough information for the other t honest replicas to finish dispersal.
- Hence, it follows that *at some point* during dispersal the message M must be held solely by $n - 2t$ honest replicas—meaning that they each hold $\frac{|M|}{n-2t}$ bits.
- But they cannot complete dispersal with this much data, or else we won't have optimal storage blowup. After assisting the honest-but-clueless replicas, they must delete data to reduce storage down to $\frac{|M|}{n-t}$ bits.

4.2 Our Initial Construction

In this section, we construct an AVID protocol that provides optimal storage against both malicious and fail-stop adversaries. That is: the construction has an optimal storage blow-up of $\frac{n}{k}$ for any choice of $k \in [t+1, n-t]$. It is *not* robust; nevertheless, we will use it as a stepping stone toward the robust construction in §5. The construction itself is detailed in Algorithm 1, and we describe it below.

Main idea. Our construction mostly follows the three-step structure of prior work stated in §4.1: we will commit to the data, reliably broadcast the commitment, and send erasure-coded fragments of the data. Additionally, we intertwine these steps carefully in order to complete dispersal in just three rounds of communication, which is known to be optimal [6]. However, our key difference is in the way that the dealing-client encodes the message and how it's being dispersed. The main message is split into erasure coded fragments of optimal size $\frac{|M|}{k}$; each fragment m_i is then split into erasure coded fragments of size $\frac{|m_i|}{n-2t}$, which we call sub-fragments and denote as $m_{i,1}, m_{i,2}, \ldots, m_{i,n}$. Each replica j receives n sub-fragments from the dealing-client, one from each fragment m_i; hence, at the moment each replica holds $\frac{|M|}{n-2t}$ bits of data. Then, each honest replica can redistribute one sub-fragment to every other replica. This redistribution achieves the goals that (i) only $n - 2t$ replicas need to redistribute data for everybody to complete dispersal with a valid fragment, and (ii) after redistribution, each replica stores only the optimal $\frac{|M|}{n-t}$ bits of data.

This construction uses vector commitments, without any additional features. They can be instantiated using Merkle trees. Note that there do exist other choices of vector commitments, some of which achieve constant-size proofs based on trusted setup, which can lower the additive overhead on the storage complexity. Two benefits of Merkle trees are that they require no setup and we can use, which do not require any setup and are amenable to a form of aggregation (based on subtree structure) to reduce storage complexity, as shown in §4.4 and Fig. 2.

Algorithmic details. Our protocol in Alg. 1 initial construction follows the structure of Bracha's reliable broadcast [6]. Our *avid-disperse* protocol has three main steps:

(1) Broadcast Phase: A dealing-client starts the dispersal phase when it wants to disperse a new message M. The dealing-client encodes the message M into fragments using erasure coding, so that any k fragments can reconstruct M. Each fragment m_i is then split into n erasure-coded sub-fragments $m_{i,j}$ such that any $n-2t$ of them can reconstruct m_i. All n^2 sub-fragments are then hashed and committed to (in a well-specified order) using a vector commitment; denote the result as c. The dealing-client then sends each replica j the following: a list $m'_j = \{m_{1,j}, \ldots, m_{n,j}\}$ that contains the j^{th} sub-fragment of every fragment, the vector commitment c, and a list π_j that contains n opening proofs corresponding to the n sub-fragments contained in m'_j.
(2) Echo Phase: Each replica j verifies using VVerify that all sub-fragments in m'_j are consistent with the same vector commitment commit using the opening proofs in π_j. If the check is valid, the honest replica sends an echo message to every replica i with its corresponding sub-fragment $m_{i,j}$, its proof $\pi_{i,j}$ and commitment c.
(3) Ready Phase: This is nearly identical to the third step of Bracha's reliable broadcast [6]. A replica sends a `READY` in two cases: (1) if it has received $n - t$ `ECHO` with valid sub-fragments and proofs for the same vector commitment c, or (2) if it received $t + 1$ `READY` with the same vector commitment c.

A replica completes *avid-deliver* and stores $n - 2t$ sub-fragments in long term storage if: it has received $n - t$ ready messages with the same c, and it has received $n-2t$ echo messages with sub-fragments and proofs that are consistent with the same vector commitment c. In case the replica received more than valid $n - 2t$ sub-fragments, it can safely delete any extra ones and only store $n - 2t$ of them along with their opening proofs.

Subsequently, *avid-retrieve* works as follows. The client invokes `RETRIEVE`, after which each replica sends back the list S of $n - 2t$ tuples acquired from the echo. The client verifies that the list S has length $n - 2t$, contains the same c, and that each sub-fragment is part of the same vector commitment c. If so, the client decodes the fragment from the sub-fragments, and stores it in the dictionary D with key c. If $D[c]$ has k fragments, the client decodes the original message from the fragments to reconstruct M'. To check that M' is valid, the client re-encodes M' into fragments and sub-fragments in the same way the dealing-client did it, and commits to all fragments in a vector commitment c'. If c' is equal to c then the client terminates with M'. Otherwise, the client terminates with $\perp$.

4.3 Communication Complexity

Dispersal. Using any erasure code of optimal size, a sub-fragment size is $\frac{|M|}{k \cdot (n-2t)} = O(\frac{|M|}{n^2})$, assuming that k is proportional to

Algorithm 1 Pseudocode for Our Initial AVID Construction

▷ **dispersal**

// the dealing-client invokes DISPERSE *(M)*

input M
let $m_1, m_2, \ldots, m_n := \mathsf{RSEnc}(M, n, k)$
let $m_{i,1}, m_{i,2}, \ldots m_{i,n} := \mathsf{RSEnc}(m_i, n, n-2t)]$ for each i
let $c = \mathsf{VCom}([(\mathsf{hash}(m_{1,1}), \mathsf{hash}(m_{1,2}), \ldots \mathsf{hash}(m_{n,n}))])$
Send (m'_j, π_j, c) to every server replica j, where $m'_j = [m_{1,j}, m_{2,j}, \ldots m_{n,j}]$ and $\pi_j = [\mathsf{VGen}(c, (i \cdot n) + 1 \text{ for } i \in (0, n-1), (i \cdot n) + 1]$ along with c

// code for server replica i, upon receiving a message that satisfies any of the conditions below
// condition 1: whether replica i's message from the dealing-client is sufficient to send an echo message
upon receiving the broadcast message (m'_i, π_i, c) from the dealing-client where the j^{th} sub-fragment $m_{j,i}$ verifies $\mathsf{VVerify}(\mathsf{hash}(m_{j,i}), \pi_{j,i}, c)$ **do**
    if replica i hasn't sent **echo** before, Send $(\textbf{echo}, m_{j,i}\pi_{j,i}, c)$ to every replica j

// condition 2: whether replica i has received enough echo messages to send a ready message
upon receiving $(n-t)$ **echo** messages with the same c from other replicas, where each message verifies the predicate $\mathsf{VVerify}(m_{i,j}, \pi_{i,j}, c)$ **do**
    if replica i hasn't sent **ready** before, send $(\textbf{ready}, c)$ to every replica j

// condition 3: whether replica i has received enough ready messages to send a ready message
upon receiving $t+1$ **ready** messages with the same c from other replicas **do**
    if replica i hasn't sent **ready** before, Send $(\textbf{ready}, c)$ to every replica j

// condition 4: whether replica i has received enough ready messages to complete avid-disperse
upon receiving $n-t$ **ready** messages with the same c from other replicas **do**
    wait until $n-2t$ **echo** messages where each sub-fragment $m_{i,j}$ verifies the predicate $\mathsf{VVerify}(\mathsf{hash}(m_{i,j}), \pi_{i,j}, c)$ and have the same c *// will receive at least $n-2t$ echo messages eventually*
    store the tuple $(m_{i,j}, \pi_{i,j}, c)$ from an arbitrary valid $n-2t$ **echo** in a list S
    avid-deliver S

▷ **retrieval**

// the retrieving-client invokes RETRIEVE
send ⟨RETRIEVE⟩ to all replicas
initialize an empty dictionary $D := \{\}$
// code for server replica i
upon receiving ⟨RETRIEVE⟩ for the first time **do**
    send S to the retrieving client
upon receiving S for the first time from replica j where $(\mathsf{length}(S) = n-2t)$ and (all tuples in S contain the same c) **do**
    if $\mathsf{VVerify}(m_{j,i}, \pi_{j,i}, c)$ is **true** for all $(m_{j,i}, \pi_{j,i}, c)$ in S **then**:
        $D[c].add(\mathsf{RSDec}(\{m_{j,i}\}_{\forall m_{j,i} \in S}))$
    if $\mathsf{length}(D[c]) == k$ **then**:
        $M' = \mathsf{RSDec}(D[c])$
        let $m'_1, m'_2, \ldots, m'_n := \mathsf{RSEnc}(M', n, k)$
        let $m'_{i,1}, m'_{i,2}, \ldots m'_{i,n} := \mathsf{RSEnc}(m'_i, n, t+1)]$ for each i
        let $c' = \mathsf{VCom}([(\mathsf{hash}(m'_{1,1}), \mathsf{hash}(m'_{1,2}), \ldots \mathsf{hash}(m'_{n,n})]$
        if $c' == c$ **then** *avid-output* (M')
        else *avid-output* $(\perp)$

n. Additionally, an inclusion proof is of size $\log(n)$ if Merkle trees are used, and a vector commitment's size is λ. The broadcast phase costs $O(|M| + \lambda n^2 \log(n))$. The dealing-client has to send each replica n sub-fragments with n proofs. The echo stage costs $O(|M| + \lambda n^2 \log(n))$. Each replica has to send one sub-fragment to every other replica along side an inclusion proof. The ready stage costs $O(\lambda n^2)$. Every replica has to send the vector commitment to the sub-fragments hashes. Thus the total communication complexity of the dispersal phase is $O(|M| + \lambda n^2 \log(n))$.

Retrieval. The retrieval cost is $O(|M| + \lambda n^2 \log n)$. (This is for the scheme as presented above without batching; if we add the batching technique described below in §4.4, then the cost is $O(|M| + \lambda n \log n)$.) Each replica has to send to the retrieving client n sub-fragments, their proofs and the vector commitment.

4.4 Storage Blowup

At the end of the dispersal, each replica i has $n-2t$ sub-fragments of optimal size $\frac{|M|}{k}$, because each sub-fragment is of size $\frac{|M|}{k \cdot (n-2t)}$. Moreover, if we instantiate our construction with Merkle trees, the opening proofs of each sub-fragment is of size $O(\lambda \log(n))$ which means that the total size stored per replica is $\frac{|M|}{k} + O(\lambda n \log n)$. Thus, the total storage blow-up is $\frac{n}{k}$, and with the additive overhead of the reliable broadcast of the commitment, the total storage complexity is $\frac{n}{k}|M| + O(\lambda n^2 \log n)$.

Reducing the storage overhead. At To reduce the storage overhead: observe that at the end of the dispersal phase, each replica has to store $n-2t$ proofs alongside the $n-2t$ sub-fragments. Depending on the vector commitment being used, this This adds an extra linear factor in the size of the witness of the vector commitment.

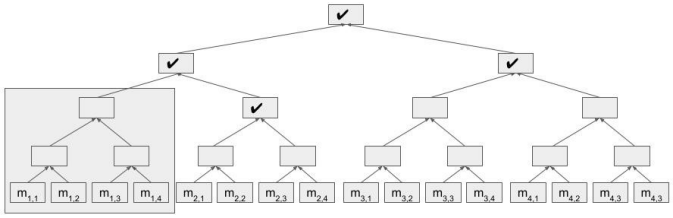

Figure 2: Batched Merkle proofs for n sub-fragments of replica number 1. Instead of having n separate Merkle proofs for each of the sub-fragments $m_{1,1}, m_{1,2}, m_{1,3}, m_{1,4}$, one proof of size $O(\lambda \cdot \log(n))$ is enough for all n of them.

By using vector aggregatable commitments, those $n - 2t$ proofs can be compressed into just one proof. Thus, in the retrieval, each replica can send its $n - 2t$ sub-fragments and only one proof to the receiver instead of $n - 2t$ proofs. In the case of Merkle trees, aggregating multiple contiguous proofs if n is a power of 2 is easy: For brevity, we describe the the idea in the case that n is a power of 2: one can send one proof for the sub-tree root, as shown in Fig. 2. Using aggregation reduces the total storage complexity to $\frac{n}{k}|M| + O(\lambda n \log n)$.

It only remains to reason about security, as shown below.

Theorem 2. *The AVID construction in Algorithm 1 construction in this section is an (n, t, k) asynchronous verifiable information dispersal AVID scheme for $t \leq \frac{n-1}{3}$ and $t + 1 \leq k \leq n - t$.*

We defer the proof to Appendix C.

Partial retrieval. To enable partial retrieval of a message block $M = M_1, \ldots, M_l$ at any index $i \leq l$, a client could run l parallel instances of the *avid-disperse* protocol for each sub-block. However, this approach would multiply both communication and storage complexities by a factor of l. In the forthcoming construction, we aim to mitigate this issue by substituting the multiplicative increase in overhead with an additive one.

5 COMMITTING AVID WITH PARTIAL RETRIEVAL

In this section, we tweak the above construction to provide robustness and partial retrieval while still achieving optimal storage blow-up. Our main idea is to instantiate the erasure code with a concrete, novel bi-dimensional erasure coding scheme, which might have independent interest.

5.1 Bi-dimensional erasure coding

A bi-dimensional erasure coding scheme is a generalized form of erasure coding (cf. §2) that simultaneously encodes a message M under two different linear erasure codes. The encode algorithm denoted $encode(M, r, k, n)$ takes a message M and outputs a matrix of n^2 fragments as depicted in Figure 3. Any set s of those fragments (and their corresponding indices within the matrix) that contains at least r rows with at least k fragments each, or at least k columns with at least r fragments each, can be used to recover M using either *row-wise-decode* or *column-wise-decode*. A (r, k, n) erasure coding scheme is a tuple of algorithms (*encode*, *row-wise-decode*,

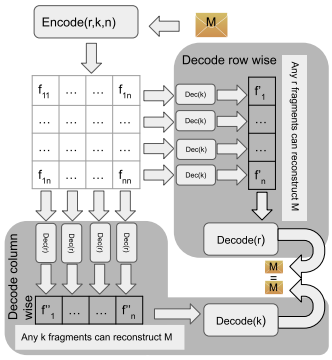

Figure 3: Overview of the bi-dimensional encoding. Message M is broken down into n^2 sub-fragments using a (r, k, n) bi-dimensional erasure coding. Any k sub-fragments on each row i, is encoding a fragment f_i' of a (r, n) uni-dimensional erasure coding for a message M. While at the same time, any r sub-fragments on each column j, is encoding a fragment f_j'' of a (k, n) uni-dimensional erasure coding of the same message M.

column-wise-decode). Here, *row-wise-decode*(r, k, s) consists of calling the uni-dimensional *decode*(k) over k fragments in r rows, followed by a *decode*(r) over the r outputs. Analogously, the algorithm *column-wise-decode*(r, k, s) consists of calling the uni-dimensional *decode*(r) over r fragments in k columns, followed by a *decode*(k) over the k outputs.

Correctness guarantee. Let $s = encode(M, r, k, n)$. Let s_1' and s_2' be two subsets of fragments in s (and their corresponding indices) such that s_1' contains at least r rows of s with at least k fragments each, and s_2' contains at least k columns in s with at least r fragments each. Then *row-wise-decode*$(r, k, s_1') =$ *column-wise-decode*$(r, k, s_2') = M$.

Reed-Solomon bi-dimensional erasure coding. In this paper, we will instantiate our bi-dimensional erasure coding using systematic Reed-Solomon erasure encoding. For a prime power q and any $n \leq q$, a (r, k, n) bi-dimensional systematic Reed-Solomon erasure code works on messages M that are $k \times r$ matrices of symbols, where each symbol is an element of the finite field GF(q). The symbols are then treated as evaluations of a bi-variate polynomial P of degree $(k - 1)$ in one variable and $(r - 1)$ in the other variable over GF(q). The encoding function, denoted RSEnc for this code, computes each fragment f_{ij} by evaluating P at different elements i and j of the GF(q) (we will assume some canonical injection of $\{1, \ldots, n\}$ into GF(q) and, slightly abusing notation, will write $P(i, j)$ for integers $i, j \in [1, n]$ assuming that i and j will get mapped to GF(q) before P is evaluated).

We will also use a partial encoding and decoding algorithms *column-RSEnc* and *column-RSDec*. *column-RSEnc* evaluates P on $1 \ldots n$ to produce n uni-variate polynomials $\phi_1, \ldots, \phi_n$ each of degree $r - 1$ (with $\phi_i(\cdot) = P(\cdot, i)$). While *column-RSDec* takes any k univariate polynomials produced by *column-RSEnc* to reconstructs P.

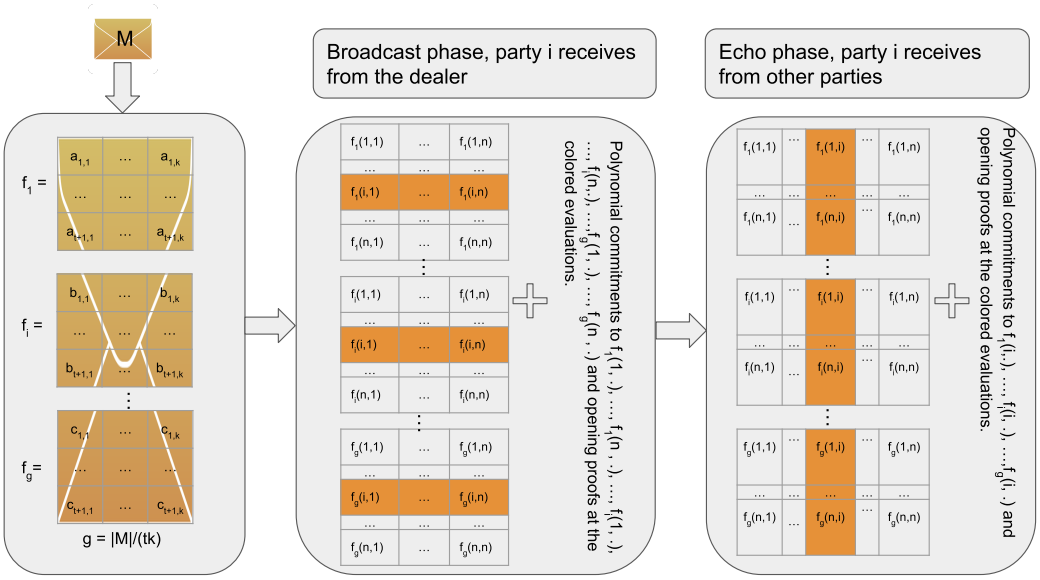

Figure 4: Overview of the dispersal and retrieval phase of our Robust AVID protocol when $n = 3t + 1$. Dispersal is done for a message M of size $\lambda(t + 1) \cdot k$.

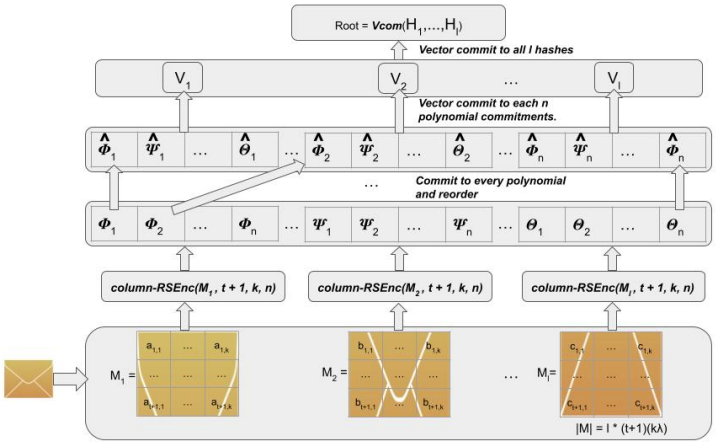

Figure 5: Overview of the root commitment when a message M is of size $l \cdot k \cdot (t + 1) \cdot \lambda$ for any positive integer l (when $n = 3t + 1$). Each message is broken into l chunks. Each chunk is then encoded into n polynomials. The polynomials are then committed to using polynomial commitments before being sorted and vector committed into n vector commitments. The n vector commitments are then committed to using a vector commitment to produce the Root vector commitment.

5.2 Our construction

Main idea. The previous construction has an optimal storage AVID that is not robust. This is due to the fact that we don't check the consistency of the fragments until the reconstruction phase. In this section, we build the first robust AVID with optimal storage blowup and communication complexity of $O(|M|)$. Starting from the ideas in §4, our main requirement is to guarantee successful retrieval even for malicious dealers.

We achieve this using bi-dimensional erasure coding based on Reed-Solomon with $r = n-2t$ rows and k columns. This lets the honest replicas check the validity of the fragments in a decentralized way before the end of the dispersal phase. We describe our construction below using KZG polynomial commitments [18], although the technique generalizes to any linear erasure coding scheme using inner product arguments including Bulletproofs [7]. We present a detailed construction in Algs. 2-3 for robust AVID with a fixed message block M of size $k \cdot (n - 2t) \cdot \lambda$. Then, we explain how it can be extended to arbitrarily large messages of size $l \cdot k \cdot (n - 2t) \cdot \lambda$ for any integer l.

Algorithmic details. Our robust AVID protocol is shown in Algorithm 2. Just like our non-robust construction from §4, it follows the 3-round structure of Bracha reliable broadcast.

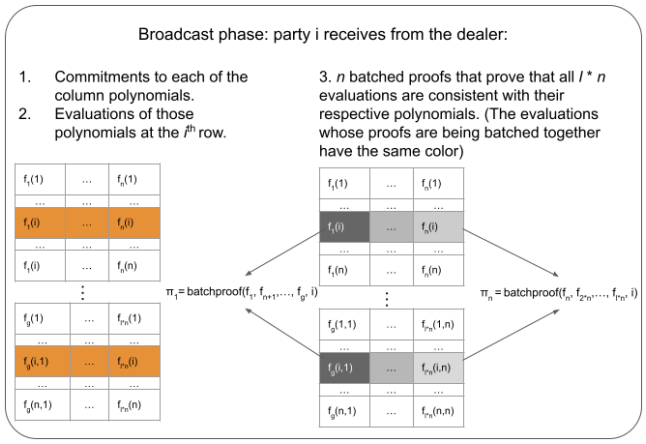

Figure 6: Overview of the broadcast phase with batching, when $n = 3t + 1$ and a message M is of size $l \cdot k \cdot (t + 1)) \cdot \lambda$ for any positive integer l.

Algorithm 2 Dispersal stage of our robust AVID, for a message M of size $k \cdot (n - 2t) \cdot \lambda$

▷ **performed by the dealing-client**

1: **input** M
2: let $\phi_1 \dots \phi_n = \textit{column-RSEnc}(M, n - 2t, k, n)$ // *each ϕ_i is a univariate polynomial of degree t*
3: Send $([\phi_1(i), \dots, \phi_n(i)], [\pi_1 = \text{open}(\text{pp}, \phi_1, i), \dots, \pi_n = \text{open}(\text{pp}, \phi_n, i)], [\hat{\phi}_1 = \text{Com}(\text{pp}, \phi_1, t), \dots, \hat{\phi}_n = \text{Com}(\text{pp}, \phi_n, t)])$ to every replica i

▷ **performed by every server replica** i

4: **upon** receiving the broadcast message $([\phi_1(i), \dots, \phi_n(i)], [\pi_1, \dots, \pi_n], [\hat{\phi}_1, \dots, \hat{\phi}_n])$ from the dealing-client where for every $j \in [1 \dots n]$, $\text{Verify}(\text{pp}, \hat{\phi}_j, \langle i, \phi_j(i), \pi_j \rangle, t) = True$ **do**
5: check that $[\phi_1(i), \dots, \phi_n(i)]$ lie on the same polynomial of degree k
6: let $H = [(\text{hash}(\hat{\phi}_1), \dots \text{hash}(\hat{\phi}_n)]$
7: if replica i hasn't sent **echo** before, send $(\textbf{echo}, \phi_j(i), \pi_j, \hat{\phi}_j, H_{\pi_j} = \text{VGen}(H, j), \hat{H} = \text{VCom}(H))$ to every replica j
8: **upon** receiving $(n-t)$ **echo** messages with the same $\hat{H}$ from other replicas, where each message satisfies the two predicates $\text{VVerify}(\text{hash}(\hat{\phi}_i), H_{\pi_i}, \hat{H})$ and $\text{Verify}(\text{pp}, \hat{\phi}_i, \langle j, \phi_i(j), \pi_j \rangle, t)$ **do**
9: if replica i hasn't sent **ready** before, send $(\textbf{ready}, \hat{H})$ to every replica j
10: **upon** receiving $(t + 1)$ **ready** messages with the same $\hat{H}$ from other replicas **do**
11: if replica i hasn't sent **ready** before, send $(\textbf{ready}, \hat{H})$ to every replica j
12: **upon** receiving $n - t$ **ready** messages with the same $\hat{H}$ from other replicas **do**
13: wait until $n - 2t$ **echo** messages with the same $\hat{H}$ where each message satisfies the two predicates $\text{VVerify}(\text{hash}(\hat{\phi}_i), H_{\pi_i}, \hat{H})$ and $\text{Verify}(\text{pp}, \hat{\phi}_i, \langle j, \phi_i(j), \pi_j \rangle, t)$ // *will receive at least $n-2t$ echo messages eventually*
14: interpolate the polynomial ϕ_i
15: *avid-deliver* the tuple $(\hat{H}, H_{\pi_i}, \phi_i)$

Algorithm 3 Partial retrieval stage for our robust AVID protocol, for a message M of size $k \cdot (n - 2t) \cdot \lambda$

// *the retrieving client invokes* RETRIEVE *on the indices of the rows it wants to fetch*
1: **send** $\langle \text{RETRIEVE}, \textit{indices} \rangle$ to all replicas
2: let $dict = \{\}$
// *code for replica i*
3: **upon** receiving $\langle \text{RETRIEVE} \rangle$ for the first time **do**
4: send $(\hat{H}, H_{\pi_i}, \hat{\phi}_i, \pi_i = \textit{open}_{agg}(\text{pp}, \phi_i, \textit{indices})$ to the retrieving client
5: **upon** receiving $(\hat{H}, H_{\pi_j}, \hat{\phi}_j, \pi_j)$ for the first time from replica j **do**
6: **if** $\text{VVerify}(\text{hash}(\hat{\phi}_j), H_{\pi_j}, \hat{H})$ and $\textit{Verify}_{agg}(\text{pp}, \hat{\phi}_j, \pi_j)$ and $\pi_j[0] == \textit{indices}$
$rows = [\ ]$
7: **for** i in range(len(indices)) **do**
8: $dict[\hat{H}][\pi_j[0][i]].add((j, \pi_j[1][i]))$
9: **if** $\textit{length}(dict[\hat{H}][\pi_j[0][i]]) == k$
10: $rows.add(decode(dict[\hat{H}][\pi_j[0][i]]))$
11: *avid-output* $rows$

(1) Broadcast Phase: A dealing-client starts the dispersal phase when it wants to disperse a new message M. The dealing-client uses a partial encoding algorithm *column-RSEnc* with $q = 2^\lambda$ (where λ is the security parameter) to produce n uni-variate polynomials $\phi_1, \dots, \phi_n$ each of degree t; any k of those polynomials can be used to generate all n polynomials by using interpolation. The dealing-client commits to each polynomial using a polynomial commitment. The dealing-client then sends each replica i, all n polynomial commitments $\hat{\phi}_1 \dots \hat{\phi}_n$ together with an evaluation of each polynomial ϕ_j at i with the proper opening proof π_j.
(2) Echo Phase: Each replica i checks using Verify that every claimed evaluation $\phi_j(i)$ is consistent with the polynomial commitment $\hat{\phi}_j$ at i using π_j. Replica i then checks that all points $(1, \phi_1(i)), \dots, (n, \phi_n(i))$ form a polynomial of degree k. If all checks are successful, the replica commits to all n polynomial commitments using a vector commitment $\hat{H}$ by calling VCom on the list of every hash of every polynomial commitment called H. The honest replica sends one echo message to every replica j containing the vector commitment $\hat{H}$, a polynomial commitment $\hat{\phi}_j$ with its proper inclusion proof H_{π_j}, and the point $\phi_j(i)$ with its opening proof π_j.
(3) Ready Phase: A replica sends a READY to every other replica in two cases: 1) if it has received $n - t$ ECHO messages (each from a different replica) with the same vector commitment $\hat{H}$ that have passed the checks on line 8 of Alg. 2) if it received $t + 1$ READY with the same vector commitment $\hat{H}$.

Finally, if replica i receives $n - t$ ready messages from different replicas with the same $\hat{H}$, then replica i terminates and stores the polynomial ϕ_i, the vector commitment $\hat{H}$, and the inclusion proof H_{π_i} in long term storage. Replica i would be able to do so because it has, or will eventually receive at least $n - 2t$ echo messages with the same $\hat{H}$ and that each has passed the two validity checks on line 13. The first validity check will ensure that $\hat{\phi}_i$ is part of the

vector commitment $\hat{H}$ and the second validity check will insure that a new point on ϕ_i is present.

Subsequently, *avid-retrieve* works as follows: the client invokes RETRIEVE while specifying which rows they wish to reconstruct. To ensure the query's relevance, we limit the focus to rows between 1 and $t+1$ because of the use of systematic erasure codes. This range is critical because, in systematic erasure codes, the first $t+1$ rows map directly to the the original data block.

Each replica i sends back the polynomial commitment $\hat{\phi}_i$ and the evaluation of ϕ at the required indices together with an aggregated proof π proving that all evaluations are correctly on the polynomial, the vector commitment $\hat{H}$, and the inclusion proof H_{π_i}. The client checks 1) that the polynomial commitment is part of the vector commitment $\hat{H}$ using the inclusion proof H_{π_i}, 2) checks that the evaluations are correct using the proof π. If all checks are valid then each point is stored in the dictionary at $\hat{H}$ at the right row index. Once a row has k elements that row is decoded using standard reed-solomon decoding.

Extending Algorithms 2-3 to support larger messages of size $l \cdot k \cdot (n-2t) \cdot \lambda$ for any $l > 1$. One can extend this AVID protocol to larger messages by running l multiple instances of the protocol in parallel on smaller chunks of the data. However, doing so naively would multiply the additive overhead for communication and storage by a factor of l, because the replicas would construct and send separate proofs and commitments for each chunk of the data.

Instead, we make use of batching (see Fig. 6) to reduce the number of proofs that need to be sent and slightly change the way we commit to the polynomial commitments (depicted in Figure 5). We adopt a recursive approach: the l polynomial commitments for a single replica are committed using one vector commitment, and H (cf. line 6 of Algorithm 2) becomes a vector commitment of the n per-replica vector commitments. This way each replica only needs to store one proof of inclusion for the l polynomials it has stored.

We discuss the full details in Appendix D.

5.3 Communication Complexity

Dispersal. Consider a message M of size $|M| = l \cdot k \cdot (n-2t) \cdot \lambda$ and for any $k \geq 1$. During dispersal: the dealing-client broadcasts $l \cdot n$ polynomial commitments and evaluations along with the corresponding opening proofs, and then each server replica sends every other replica l polynomial commitments and evaluations in the echo stage and one vector commitment in the ready stage. Table 1 shows the communication complexity when these primitives are instantiated with Bulletproofs or KZG commitments. We measure the communication for each stage of dispersal.

- Broadcast: the dealing-client sends each replica $l \cdot n$ polynomial commitments and $l \cdot n$ evaluations along with the corresponding (batched, if $l > 1$) opening proofs. Note that the $l \cdot n$ evaluations have a cumulative size of $O(\frac{|M|}{n})$, if t is proportional to n.
- Echo: each replica sends every replica l polynomial commitments, l evaluations with its proper one batched proof, a vector commitment, and one vector commitment opening proof. Note that the l evaluations cost $O(\frac{|M|}{n^2})$ if k is proportional to n.
- Ready: each replica sends every other replica a vector commitment of size λ, for a total of $O(\lambda n^2)$ communication.

If KZG commitments [18] are used to instantiate both the polynomial and vector commitment schemes, then this would result in a polynomial and vector commitment size of $O(\lambda)$, a batch proof size of $O(\lambda)$ [2] and a vector commitment opening proof of also $O(\lambda)$. As a result, the total total communication complexity would be $O(|M| + \lambda n^2)$. If instead one instantiates the polynomial commitment with Bulletproofs [7] and the vector commitment with Merkle trees, the resulting scheme no longer needs trusted setup but the (batched) proofs are each of size $O(\lambda \log n)$ so the communication complexity would be $O(|M| + \lambda n^2 \log n)$.

Partial retrieval. Consider a retrieving client with index i and a message M, where $|M| = k \cdot (n-2t) \cdot \lambda$ and for any $k = O(n)$. Each replica j sends its own polynomial evaluation at i, the corresponding polynomial commitment, and the inclusion proof of the polynomial in the vector commitment. Hence, the complexity to retrieve one row is $O(\lambda n)$ with KZG commitments [18], or $O(\lambda n \log n)$ with Bulletproofs [7].

5.4 Storage Blowup

For a message of size $|M| = k \cdot (n-2t) \cdot \lambda$, each server replica i must store its univariate column polynomial ϕ_i with $n-2t$ coefficients (which requires $\frac{|M|}{k}$ storage), along with the vector commitment H (cf. line 6 of Algorithm 2) and proof that its polynomial commitment $\hat{\phi}_i$ is contained in H. This vector commitment and proof are $O(\lambda)$-sized if KZG commitments are used, and the proof is $O(\lambda \log n)$ if bulletproofs are used. Across all n replicas, the storage blowup is $\frac{n}{k}$ and the total storage cost is shown in Table 1.

For larger messages of size $|M| = l \cdot k \cdot (n-2t) \cdot \lambda$: the storage blowup is still $\frac{n}{k}$, but if done naively with a separate commitment and proof for each chunk then the replicas would need l times as much storage to hold all the proofs. By using the recursive approach described in §5.2, we can improve the storage costs to again match Table 1.

Theorem 3. *The construction in Algorithms 2-3 is an (n, t, k) robust and committing AVID for $t+1 \leq k \leq n-t$ and $t \leq \frac{n-1}{3}$ with both optimal storage blowup and communication complexity.*

We prove this theorem in Appendix E.

ACKNOWLEDGMENTS

This material is based on work supported by DARPA under Agreement No. HR00112020021 and HR00112020023 and by the National Science Foundation under Grants No. 1801564, 1915763, 2209194, 2217770, and 2228610. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

REFERENCES

[1] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. 2022. Brief Announcement: Asynchronous Verifiable Information Dispersal with Near-Optimal Communication. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing* (Salerno, Italy) *(PODC'22)*. Association for Computing Machinery, New York, NY, USA, 418–420. https://doi.org/10.1145/3519270.3538476

[2] Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. 2021. Succinct Erasure Coding Proof Systems. Cryptology ePrint Archive, Report 2021/1500. https://eprint.iacr.org/2021/1500.
[3] Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. 2022. Practical and Improved Byzantine Reliable Broadcast and Asynchronous Verifiable Information Dispersal from Hash Functions. Cryptology ePrint Archive, Report 2022/171. https://eprint.iacr.org/2022/171.
[4] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. 2021. High-Threshold AVSS with Optimal Communication Complexity. In *FC 2021, Part II (LNCS, Vol. 12675)*, Nikita Borisov and Claudia Díaz (Eds.). Springer, Heidelberg, 479–498. https://doi.org/10.1007/978-3-662-64331-0_25
[5] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM CCS 93*, Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby (Eds.). ACM Press, 62–73. https://doi.org/10.1145/168588.168596
[6] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Information and Computation* 75, 2 (1987), 130–143.
[7] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 315–334. https://doi.org/10.1109/SP.2018.00020
[8] Christian Cachin and Stefano Tessaro. 2005. Asynchronous verifiable information dispersal. In *SRDS*. IEEE, 191–201.
[9] Dario Catalano and Dario Fiore. 2013. Vector Commitments and Their Applications. In *PKC 2013 (LNCS, Vol. 7778)*, Kaoru Kurosawa and Goichiro Hanaoka (Eds.). Springer, Heidelberg, 55–72. https://doi.org/10.1007/978-3-642-36362-7_5
[10] Ethan Cecchetti, Ben Fisch, Ian Miers, and Ari Juels. 2019. PIEs: Public Incompressible Encodings for Decentralized Storage. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 1351–1367. https://doi.org/10.1145/3319535.3354231
[11] Ivan Damgård, Chaya Ganesh, and Claudio Orlandi. 2019. Proofs of Replicated Storage Without Timing Assumptions. In *CRYPTO 2019, Part I (LNCS, Vol. 11692)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Heidelberg, 355–380. https://doi.org/10.1007/978-3-030-26948-7_13
[12] Sourav Das, Zhuolun Xiang, and Ling Ren. 2021. Asynchronous Data Dissemination and its Applications. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, 2705–2721. https://doi.org/10.1145/3460120.3484808
[13] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew K. Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical Asynchronous Distributed Key Generation. In *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2518–2534. https://doi.org/10.1109/SP46214.2022.9833584
[14] Ben Fisch. 2018. PoReps: Proofs of Space on Useful Data. Cryptology ePrint Archive, Report 2018/678. https://eprint.iacr.org/2018/678.
[15] Ben Fisch. 2018. Tight Proofs of Space and Replication. Cryptology ePrint Archive, Report 2018/702. https://eprint.iacr.org/2018/702.
[16] James Hendricks, Gregory R. Ganger, and Michael K. Reiter. 2007. Verifying distributed erasure-coded data. In *26th ACM PODC*, Indranil Gupta and Roger Wattenhofer (Eds.). ACM, 139–146. https://doi.org/10.1145/1281100.1281122
[17] Ari Juels and Burton S. Kaliski Jr. 2007. Pors: proofs of retrievability for large files. In *ACM CCS 2007*, Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson (Eds.). ACM Press, 584–597. https://doi.org/10.1145/1315245.1315317
[18] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT 2010 (LNCS, Vol. 6477)*, Masayuki Abe (Ed.). Springer, Heidelberg, 177–194. https://doi.org/10.1007/978-3-642-17373-8_11
[19] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. 2020. Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 1751–1767. https://doi.org/10.1145/3372297.3423364
[20] Benoît Libert and Moti Yung. 2010. Concise Mercurial Vector Commitments and Independent Zero-Knowledge Sets with Short Proofs. In *TCC 2010 (LNCS, Vol. 5978)*, Daniele Micciancio (Ed.). Springer, Heidelberg, 499–517. https://doi.org/10.1007/978-3-642-11799-2_30
[21] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew K. Miller. 2019. HoneyBadgerMPC and AsynchroMix: Practical Asynchronous MPC and its Application to Anonymous Communication. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 887–903. https://doi.org/10.1145/3319535.3354238
[22] Kamilla Nazirkhanova, Joachim Neu, and David Tse. 2021. Information Dispersal with Provable Retrievability for Rollups. In *Advances in Financial Technologies (AFT)*. ACM.
[23] Michael O Rabin. 1990. The information dispersal algorithm and its applications. In *Sequences*. Springer, 406–419.
[24] Irving S. Reed and Gustave Solomon. 1960. Polynomial Codes Over Certain Finite Fields. *Journal of The Society for Industrial and Applied Mathematics* 8 (1960), 300–304.
[25] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. 2022. DispersedLedger: High-Throughput Byzantine Consensus on Variable Bandwidth Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.
[26] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. 2022. hbACSS: How to Robustly Share Many Secrets. In *NDSS*. The Internet Society.

A COMMUNICATION LOWER BOUND

To define a lower bound on communication, we define a *communication blow-up* in similar spirit to the storage blow-up. This is done so that we consider the case where a partial message is "big enough" to reconstruct the original message. Given a constant size commitment, a replica with enough time can compute a full message. Therefore, we restrict the replicas to a run time that is bounded by a polynomial time machine that is polynomial in the security parameter λ. The key point is that λ is independent from the message size. We think that this is a good enough model, because if the replicas can compute "big enough" messages from a commitment, then there is no point in sending the original message. The commitment itself should be the message.

Definition 4. The **communication blow-up** of a message M is the ratio of the communication complexity of the dispersal protocol and $|M|$. The overall communication blowup of an information dispersal scheme is the maximum (over the choice of $M \in \mathcal{M}$ and adversarial behavior) of the communication complexity of honest replicas, in the limit as $|M| \to \infty$.

In this section, we prove the following theorem.

Theorem 4. *Every (n, t, k) AVID protocol that satisfies Def. 1 must have a communication blow-up of at least $\max\{\frac{n}{k}, \frac{n}{n-2t}\}$ (or equivalently, $\frac{n}{\min\{k, n-2t\}}$).*

We split the proof of this theorem into Lemmas 1 and 2.

Lemma 1. *Every (n, t, k) AVID protocol A that satisfies Def. 1 must have a communication blow-up of at least $\frac{n}{k}$.*

This lemma is an immediate consequence of the fact that every (n, t, k) AVID protocol has storage blowup of at least $\frac{n}{k}$, as stated by Cachin and Tessaro [8] without proof. We provide a rigorous proof, both for completeness and because the same ideas will also be used in our proof of Lemma 2.

Proof. Assume that the adversary never corrupts any protocol participant (i.e., the dealing-client, retrieving-client, and all replicas behave honestly), and the adversary permits the network to deliver all protocol messages in some fixed order. Fix a replica i. Imagine executing the AVID protocol A for all messages $M \in \{0,1\}^\ell$ and observing the *view* of replica i (defined as the messages that the replica received from others, together with message origin information and sequence number per originator). Let p_i be the number of distinct views observed. Without loss of generality, order the replicas by p_i, lowest to highest, so that $p_1 \le \cdots \le p_n$. Note that the expected number (over all messages) of bits sent to replica i is at least $\log_2 p_i$, and thus the total expected communication (by linearity of expectation) is at least $\sum_{i=1}^{n} \log_2 p_i$.

By availability and correctness of the AVID protocol A, all $M \in \{0,1\}^\ell$ will be reconstructed correctly from the joint views of any k replicas—in particular, from the first k replicas. Since the number of

distinct joint views of the first k replicas is at most $\prod_{i=1}^{k} p_i$, and each joint view reconstructs only one message, we have $\prod_{i=1}^{k} p_i \geq 2^\ell$. Therefore, taking the logarithm of both sides, $\sum_{i=1}^{k} \log_2 p_i \geq \ell$, and thus $\log_2 p_k \geq \ell/k$. Thus, for each replica i from the remaining $n-k$ replicas, $\log_2 p_i \geq \log_2 p_k \geq \ell/k$. Thus, the total communication is

$$\begin{aligned}\sum_{i=1}^{n} \log_2 p_i &= \sum_{i=1}^{k} \log_2 p_i + \sum_{i=k+1}^{n} \log_2 p_i \\ &\geq \ell + (n-k) \cdot \ell/k \\ &= n\ell/k.\end{aligned}$$

Since the message length is ℓ, it follows that the communication blowup is n/k. □

Lemma 2. *Every (n, t, k) AVID protocol A that satisfies Def. 1 must have a communication blow-up of at least $\frac{n}{n-2t}$.*

Proof. Define p_i and the ordering i of replicas in the same way as in the proof of Lemma 1. Concretely, p_i equals the number of distinct views observed by replica i when the AVID protocol A is executed over all messages $M \in \{0,1\}^\ell$ and all replicas behave honestly, and the replicas are ordered from smallest to largest p_i.

Now, consider the execution of AVID protocol A (on any message $M \in \{0,1\}^\ell$) in two not-fully-honest worlds that we describe below. In World 1 the dealing-client is honest, and in World 2 the dealing-client is dishonest. In both worlds, we partition the replicas into three groups:

- Replicas 1 through $n-2t$ are honest throughout the protocol. We call them the "informed" replicas. Looking ahead, much of our analysis of *avid-disperse* will be from the perspective of these replicas.
- The next t replicas will communicate with the informed replicas, and for this reason we refer to this set as the "assisting" replicas.
- The final t replicas will be constructed in such a way that they don't communicate with the informed replicas; hence, we call them the "isolated" replicas.

In World 1 the dealing-client is honest and follows the protocol honestly. However, the adversary lets the t corrupted replicas fail, and as a result they will not interact with any other replicas or clients throughout *avid-disperse* and *avid-retrieve*.

In World 2 the adversary controls the dealing-client, the assisting replicas, and the network. In more detail:

- The adversary corrupts the dealing-client and assisting replicas. They behave honestly in all communications with the informed replicas and (other) assisting replicas, but they do not send any messages to the isolated replicas. Additionally, the assisting replicas do not participate in *avid-retrieve*.
- Using its control of the network, the adversary delays any messages sent from the isolated replicas to the informed replicas until after the informed replicas complete *avid-disperse* (but messages sent by the informed or assisting replicas are never delayed).

We argue that the AVID protocol terminates in both worlds. In World 1, termination is guaranteed by Def. 1 since the dealing-client is honest. In World 2, we cannot directly call upon the termination property from Def. 1 since the dealing-client is malicious. Nevertheless, we observe that the informed replicas must also complete *avid-deliver* in World 2, since they cannot distinguish between the two worlds (specifically, their views consist of honest messages from a set of $n-t$ replicas in both worlds). As a result, World 2 should also meet all the properties of AVID (including retrieval consistency and availability) even though the dealing-client is acting in a malicious way by withholding shares from the isolated replicas.

We now analyze World 2 in more detail. As in the proof of Lemma 1, we focus on the view of the (honest and) informed replicas across 2^ℓ invocations of the AVID protocol A for all possible messages $M \in \{0,1\}^\ell$. We define p_i^* as the number of distinct views observed by replica i during protocol A in World 2 with this malicious adversary, up to the point that all informed replicas complete *avid-disperse*. Note that p_i^* may be different from p_i (the number of views in the honest execution of AVID protocol A). We make two observations.

(1) For an informed replica i, its view during the execution of *avid-disperse* in World 2 is a subset of its view during a fully honest execution of *avid-disperse*; that is, $p_i^* \leq p_i$. This is because an informed replica i (in World 2) can only expect to receive messages from $n-t$ replicas, and the other t replicas either never send a message or have their messages indefinitely delayed.
(2) Let V denote the joint view of all informed replicas. For any isolated replica, its view is a deterministic function of V, since the isolated replicas only receive messages from the informed replicas, and (since A is deterministic) these messages must be a function of the informed replicas' own views. As a consequence, the joint view of all informed and isolated replicas is also a function of V.

Finally, by availability and correctness of the AVID protocol A, all $M \in \{0,1\}^\ell$ will be reconstructed correctly from the joint views of any k of the honest $n-t$ replicas during *avid-disperse*, which itself can be reconstructed from the joint view V of the informed $n-2t$ replicas.

It follows by a counting argument that $2^\ell \leq \prod_{i=1}^{k} p_i^* \leq \prod_{i=1}^{k} p_i$. Taking the logarithm of both sides yields $\sum_{i=1}^{n-2t} \log_2 p_i \geq \ell$. From the ordering of replicas i, it follows that for any $i \geq n-2t$ that $\log_2 p_i \geq \log_2 p_{n-2t} \geq \frac{\ell}{n-2t}$. Thus, the total communication is

$$\begin{aligned}\sum_{i=1}^{n} \log_2 p_i &= \sum_{i=1}^{n-2t} \log_2 p_i + \sum_{i=n-2t+1}^{n} \log_2 p_i \\ &\geq \ell + 2t \cdot \frac{\ell}{n-2t} \\ &= \frac{n\ell}{n-2t}.\end{aligned}$$

Hence, the communication blowup is $\frac{n}{n-2t}$. □

B PROOF OF THEOREM 1

In this section, we prove Theorem 1 stating that any AVID construction without a PKI must have $k \geq t+1$.

Proof. Our proof proceeds by contradiction. Let P be an arbitrary AVID protocol with $k \leq t$ that meets Definition 1. Let p_1 be an honest dispersing client who called *avid-disperse*(M) and dispersed M correctly to all n replicas. From the assumption that

$k \leq t$, the honest replicas do not have a majority of the replicas who are online and participating in the retrieval phase. Even in this scenario, the availability property would still allow any client to fetch a message because at least k replicas are present. However, the protocol P wouldn't be able to meet the correctness property because no honest majority is present.

To see why this is the case, let c be a new fetching client that is trying to reconstruct M and let A be an attacker that controls t byzantine replicas and d fail stop replicas. At the end of the dispersal phase, the attacker does the following steps: 1) turns off d honest replicas 2) deletes the state of k online malicious replicas (which is possible because $k \leq t$) 3) runs *avid-disperse*(M') (using the same avid protocol P) among these k malicious replicas it controls and $n - k$ simulated replicas. (Note that this simulation is possible because replicas have no secrets; on the other hand, if we had a PKI or some other set up, then the replicas would have secret keys corresponding to the public keys that would be known to clients, and the adversary would be unable to simulate them.)

Once c initiates *avid-retrieve*, by the correctness property as applied to *avid-disperse*(M), c will eventually output M. On the other hand, by the correctness property as applied to *avid-disperse*(M'), if c talks only to the k malicious replicas who participated in *avid-disperse*(M') (since availability requires only k honest replicas, the attacker honestly ran *avid-disperse*(M'), and the k malicious replicas act honestly with respect to *avid-disperse*(M'), correctness still applies in this case), c will eventually output M'. This is a contradiction. □

C PROOF OF THEOREM 2

Termination and Correctness. If an honest replica initiates *avid-disperse* message M then all honest replicas will pass the check VVerify($m'_{i,j}$, $\pi_{i,j}$, $\hat{H}$) because of the correctness property of the vector commitments and the encoding algorithm. Hence, $n - t$ honest replicas will send correct ECHO messages with the same $\hat{H}$ and with valid erasure coded fragments and proofs. Hence, every honest replica will eventually receive at least $n - t$ correct ECHO from the honest replicas and will send a READY message with the same $\hat{H}$. Therefore, every honest replica will eventually receive at least $n - t$ READY messages with the same $\hat{H}$ and have enough good ECHO messages to decode their own share.

Dispersal Agreement. For a replica to finish dispersal it needs to (1) receive $n - t$ READY messages with the same $\hat{H}$ and (2) receive $n - 2t$ ECHO messages with valid sub-fragments that are consistent with $\hat{H}$. If one honest replica finish the dispersal with $\hat{H}$, it is easy to see that condition 1) is met (because we are doing Bracha reliable broadcast over $\hat{H}$). We show here why condition 2 is met as well: If an honest replica completes *avid-deliver* then this means that the replica received $n - t$ READY messages where at least $n - 2t$ must have come from honest replicas. This implies that that there exist at least one honest replica that has sent a READY message with $\hat{H}$ because it received $n - t$ ECHO messages with the same $\hat{H}$, this means that $n - 2t$ honest replicas has sent $\hat{H}$ with consistent sub-fragments to all honest replicas. Therefore every honest replica will at least receive $n - 2t$ sub-fragments that are linked with $\hat{H}$ that are coming from those honest replicas.

Availability. If k honest replicas have finished dispersal this means that each replica has at least $n - 2t$ sub-fragments that are consistent with $\hat{H}$ along with the corresponding proof. Each honest replica can send the $n - 2t$ sub-fragments along side $\hat{H}$ and the proper corresponding opening proof. The retrieving client can then decode the message M from the sub-fragments.

Retrieval Consistency. Retrieving consistency is guaranteed because every retrieving client can detect an inconsistent sharing and revert the same default sharing (the message $\perp$). The detection is possible because (1) all honest replicas have consensus over the vector commitment $\hat{H}$ of all sub-fragments, (2) the reconstruction threshold is greater than the number of malicious replicas, and (3) the correctness of the decoding algorithm and the binding property of the vector commitment.

D BATCHED PROOFS FOR LONGER MESSAGES

In this section, we describe the changes needed to make Algorithm 2 support larger messages.

- In the broadcast phase on line 2, instead of running the encoding on the whole message, the message is first split into l pieces each of size $k \cdot (n - 2t) \cdot \lambda$ as depicted in Figure 5, followed by l executions of *column-RSEnc*($M_i, n - 2t, k, n$) on every smaller message M_i. This would generate in total a matrix of polynomials

$$\Phi = \begin{bmatrix} \phi_{1,1} & \dots & \phi_{1,n} \\ & \ddots & \\ \phi_{l,1} & \dots & \phi_{l,n} \end{bmatrix}.$$

  Each row i of the matrix would be filled by the output of *column-RSEnc*($M_i, n - 2t, k, n$). Just like before, the dealing-client commits to each polynomial using a polynomial commitment. The dealing-client sends each replica i, all polynomial commitments together with an evaluation of every polynomial at i. However, instead of sending one proof for every evaluation, the dealing-client batches every l evaluations on every column into one proof as depicted in Figure 6. More formally, all evaluations at i of every polynomial on column j are batched together into one proof $\pi_j = open_{batch}(\mathsf{pp}, [\phi_{1,j}, \dots, \phi_{l,j}], i)$.
- In the echo phase, just like before, every replica i checks that the evaluation that it received on every polynomial is consistent. However, $verify_{batch}$ has to be used instead of Verify. For every column j of polynomial commitments $\hat{\phi}_{1,j}, \dots, \hat{\phi}_{l,j}$, it is the case that $verify_{batch}(\mathsf{pp}, [\hat{\phi}_{1,j}, \dots, \hat{\phi}_{l,j}], \langle i, [\phi_{1,j}(i) \dots \phi_{l,j}(i), \pi_j \rangle, t)$ should be True. Also, every row r of evaluations, $\phi_{r,1}(i) \dots \phi_{r,n}(i)$ must form a polynomial of degree k. If all checks are successful, every column j (there are n columns) consisting of l polynomial commitments $\hat{\phi}_{1,j} \dots \hat{\phi}_{l,j}$ are committed to using a vector commitment to produce V_j). The replica commits to all n vector commitments using a vector commitment $\hat{H}$ by calling VCom on the list $H = V_1 \dots V_n$ as depicted in figure 5. The honest replica sends one ECHO message to every replica

j containing the vector commitment $\hat{H}$, l polynomial commitments $\hat{\phi}_{1,j} \dots \hat{\phi}_{l,j}$ with its proper inclusion proof H_{π_j}, and l points $\phi_{1,j}(i) \dots \phi_{l,j}(i)$ with its batch opening proof π_j as depicted in Figure 6.

- In the ready stage, Replica i behaves as before except that it needs to extend the checks to cater for l polynomials and save l polynomials. This is done by replacing VVerify(hash$(\hat{\phi}_i), H_{\pi_i}, \hat{H})$ with VVerify(VCom([(hash$(\hat{\phi}_{1,i}), \dots$ hash$(\hat{\phi}_{l,i}))$]), $H_{\pi_i}, \hat{H}$) and by replacing Verify(pp, $\hat{\phi}_i, \langle j, \phi_i(j), \pi_j \rangle, t)$ with $verify_{batch}$(pp, $[\hat{\phi}_{1,i}, \dots \hat{\phi}_{n,i}], \langle j, [\phi_{1,i}(j) \dots \phi_{n,i}(j)], \pi_j \rangle, t)$. Also, instead of interpolating one polynomial ϕ_i it has to interpolate l of them: $\phi_{1,i}, \dots, \phi_{l,i})$.
- In the retrieval, each replica j has the vector commitment $\hat{H}$, the vector commitment opening proof H_{π_j} (the proof is for l polynomial commitments, instead of one) and l polynomials $(\phi_{1,i}, \dots, \phi_{l,i})$ instead of one polynomial.
- For full retrieval, a fetching client, whenever they receive a message from a replica, the client has to generate l polynomial commitments from the l polynomials received, commit to them using a vector commitment and check whether they are consistent with the root vector commitment $\hat{H}$. If the check is successful, the l polynomials are added into the dictionary. After the dictionary has k values, the client can decode the original message using *column-RSDec* to recover every chunk of the message. For partial retrieval, a fetching client can specify to open a set of specific rows in any specific set of bivariate polynomials. In particular for a specific row z of a bivariate polynomial i, the fetching client can ask for k openings on k different polynomials at z with their proper polynomial commitment and proof and interpolate row z. The fetching client can verify that each polynomial commitment and corresponding polynomial evaluation at z is coming from replica j correctly. This is because each polynomial commitment must be opened at i in its corresponding vector commitment from j.

E PROOF OF THEOREM 3

In this appendix, we prove Theorem 3 that the construction in Algorithms 2-3 is a robust and committing AVID. We do so via a series of lemmas.

Lemma 3. *For any message M of size $k \cdot (n-2t)\lambda$, if an honest client initiates avid-disperse(M), then every honest replica p_i will eventually avid-deliver($\hat{H}, H_{\pi_i}, \phi_i$) where $\phi_1 \dots \phi_n$ = column-RSEnc($M, n-2t, k, n$) and $\hat{H}$ = VCom([(hash$(\hat{\phi}_1), \dots$ hash$(\hat{\phi}_n)$]). As a result, the AVID construction in Algorithm 2 satisfies the termination property in Definition 1.*

Proof. If an honest replica initiates *avid-disperse* (M) then it will create the univariate polynomials $\phi_1 \dots \phi_n$ each of degree t correctly, by calling *column-RSEnc*($M, n-2t, k, n$) and it will send to every replica i the same polynomial commitments $\hat{\phi}_1 \dots \hat{\phi}_n$ along side correct evaluations $\phi_1(i) \dots \phi_n(i)$ and valid corresponding opening proofs $\pi_1 \dots \pi_n$. The honest replicas will eventually pass the two checks on line 4 and line 5. Check 1 will pass because of the correctness property of the polynomial commitment. Check 2 will pass because of the correctness of the bi-dimensional Reed-Solomon encoding. Hence, every honest replica p_i will generate for every replica j the same $\hat{H}$ = VCom([(hash$(\hat{\phi}_1), \dots$ hash$(\hat{\phi}_n)$]), the same polynomial commitment $\hat{\phi}_j$ and the same inclusion proof H_{π_j} for $\hat{\phi}_j$. This is because both the vector and polynomial commitments being used are deterministic. p_i will send a valid ECHO message to every replica j containing $\phi_j(i), \pi_j, \hat{\phi}_j, H_{\pi_j}, \hat{H}$. Therefore, all honest replicas will receive at least $n-t$ ECHO messages with the same $\hat{H}$, and will pass the two checks on line 8. Thus, every honest replica p_i will eventually send a READY message with the same $\hat{H}$. Hence, every honest replica p_i will eventually receive at least $n-t$ READY messages with the same $\hat{H}$. Given the binding and correctness property of both the vector commitment and the polynomial commitment and the fact that that $n-t$ honest replicas sent correct ECHO messages to everyone. Replica p_i can reconstruct the polynomial ϕ_i of degree $n-2t-1$ using $n-2t$ different evaluations contained in $n-2t$ different ECHO messages. Hence, p_i will eventually terminate and call *avid-deliver*($\hat{H}, H_{\pi_i}, \phi_i$). □

Lemma 4. *Let $\phi_1 \dots \phi_n$ be a list of column polynomials of degree t. Suppose there exists a set $S \subset \{1, \dots, n\}$ of size $t+1$ such that for all $i \in S$, the row polynomial formed by interpolating $\phi_1(i), \phi_2(i), \dots, \phi_n(i)$ is of degree $k-1$. Then, there exists a unique bivariate polynomial $f(x, y)$ of degree $k-1$ in one dimension and t in the other dimension where $\phi_i(.) = f(., i)$.*

Proof. Let $S = \{x_1, \dots, x_{t+1}\}$. Let $\psi_{x_1} \dots \psi_{x_{t+1}}$ be the row polynomials of degree $k-1$ given by the statement of the lemma. Define $f(x, y) = \sum_{i=1}^{t+1} \psi_{x_i}(y) L_i(x)$, where $L_i(x)$ the appropriate Lagrange coefficient (namely, the unique degree-t univariate polynomial that vanishes at x_j for $i \neq j$ and is 1 at x_i).

Observe that f is of degree k in one variable and t in the other. Now we need to prove $f(x, y) = \phi_y(x)$. If $x \in S$, then this is true by construction, because $f(x, y) = \psi_x(y)$ (because there is only one Lagrange coefficient that doesn't vanish at x), which is equal to $\phi_y(x)$ by definition of ψ. Since $f(\cdot, y)$ and $\phi_y(\cdot)$ are degree-t polynomials that agree on $t+1$ points (namely, all points in S), they must be equal as polynomials, and thus the statement is true for all x, not just $x \in S$.

We now need to prove uniqueness. Observe that for every i, it is the case that $f(x, i)$ (as a polynomial of degree less than n in x) is unique if it agrees with $\phi_i(x)$ in n points (because two different univariate polynomials of degree less than n cannot agree on n points). Thus, viewing $f(x, y)$ as a univariate polynomial in y whose coefficients are polynomials in x, we know that its n evaluations are unique. Since it has degree less than n in y, it must also be unique. □

Lemma 5. *If an honest replica p_i calls avid-deliver($\hat{H}, H_{\pi_i}, \phi_i$), then every honest replica p_j will eventually receive at least $n-2t$ ECHO messages from $n-2t$ different replicas with the following properties:*

(1) All ECHO messages contain the same $\hat{H}$, $\hat{\phi}_j$ and H_{π_j} such that VVerify(hash$(\hat{\phi}_j)$, H_{π_j}, $\hat{H}$) *is True.*
(2) If p_s is the sender of the ECHO message, then the ECHO message also contains $\phi_j(s)$ such that Verify(pp, $\hat{\phi}_j, \langle s, \phi_j(s), \pi_s \rangle, t$) *is True.*

Proof. If an honest replica calls *avid-deliver*$(\hat{H}, H_{\pi_i}, \phi_i)$, then it must have received $n-t$ READY messages with the same $\hat{H}$, where at least $t+1$ of those READY messages must have been sent by honest replicas. Hence, there is at least one honest replica p_i that received $n-t$ ECHO messages with the same $\hat{H}$, $\hat{\phi}_i$ and H_{π_i}. Since t is the total number of malicious replicas, at least $n-2t$ replicas have each received a valid broadcast message from the dealing-client with the same n univariate polynomials $\phi_1 \dots \phi_n$ such that $\hat{H} = \text{VCom}([(\text{hash}(\hat{\phi}_1), \dots \text{hash}(\hat{\phi}_n)])$ (due to the binding property of the vector commitment). Also, those broadcast messages must have passed the two checks at line 4 and 5 respectively. The first check insures that each replica p_j as a one evaluation at j from every univariate polynomial of degree t. The second check insures that those points lie on the a degree k polynomial. Looking at Figure 4, this corresponds to every replica verifying that it's row lies on a polynomial of degree k. Taking those two checks together and the fact that $n-2t$ replicas are doing it for all n univariate polynomials then by Lemma 4 we can conclude that $\hat{H}$ is actually a commitment to a bivariate polynomial and that any k of those polynomials is enough to generate all other n polynomials. Since every replica p_s of those $n-2t$ replicas have each received a valid *BROADCAST* and passed the two checks this means that they will send every replica p_j a valid ECHO message that contains 1) the same $\hat{H}$, $\hat{\phi}_j$ and H_{π_j} such that VVerify(hash($\hat{\phi}_j$), H_{π_j}, $\hat{H}$) is $True$ and $\phi_j(s)$ such that Verify(pp, $\hat{\phi}_j$, $\langle s, \phi_j(s), \pi_s \rangle$, t) is $True$. □

Lemma 6. *No two honest replicas will receive $t+1$ READY messages with different $\hat{H}$.*

Proof. Assume two honest replicas p_1 and p_2 received two READY messages with different commitments $\hat{H}_1$ and $\hat{H}_2$. This means that p_1 either heard $n-t$ ECHO messages containing $\hat{H}_1$ or heard $t+1$ READY messages containing $\hat{H}_1$ where at least one honest replica has heard $n-t$ ECHO messages containing $\hat{H}_1$. Similarly, p_2 either heard $n-t$ ECHO messages containing $\hat{H}_2$ or heard $t+1$ READY messages containing $\hat{H}_2$ where at least one honest replica has heard $n-t$ ECHO messages containing $\hat{H}_2$. This means that there is at least two sets of replicas A and B, each of size $n-t$ where A sent an ECHO message containing $\hat{H}_1$ and B sent an ECHO message containing $\hat{H}_2$. Since $A \cap B \geq t+1$, this means that there is at least $t+1$ replicas that sent ECHO messages with different commitments. since t is the total number of bad replicas then there is at least one honest replica that sent two different ECHO messages, one containing $\hat{H}_1$ and another containing $\hat{H}_2$. This is a contradiction. □

Lemma 7. *If an honest replica p_i calls avid-deliver$(\hat{H}, H_{\pi_i}, \phi_i)$, then every honest replica will eventually receive at least $n-t$ READY messages from $n-t$ different replicas with the same $\hat{H}$.*

Proof. If an honest replica p_i calls *avid-deliver*$(\hat{H}, H_{\pi_i}, \phi_i)$ then this means that the replica heard $n-t$ READY messages coming from $n-t$ different replicas with the same $\hat{H}$. Since t is the total number of malicious replicas then $t+1$ READY messages must have came from $t+1$ different honest replicas. Since those $t+1$ will send a READY with $\hat{H}$ to everyone, then every other honest replica that didn't send a READY yet with $\hat{H}$, will also send a READY with $\hat{H}$ because of the amplification step at 9. Because there is at least $n-t$ honest replicas and because of lemma 6, all honest replicas will eventually receive at least $n-t$ READY messages from $n-t$ different replicas with the same $\hat{H}$. □

Lemma 8. *If an honest replica p_i calls avid-deliver$(\hat{H}, H_{\pi_i}, \phi_i)$, then every honest replica p_j eventually calls avid-deliver$(\hat{H}, H_{\pi_j}, \phi_j)$ such that* VVerify*(*hash$(\hat{\phi}_j)$, H_{π_j}, $\hat{H}$*) is $True$. As a result, the AVID construction in Algorithm 2 satisfies the dispersal agreement property in Definition 1.*

Proof. This property follows directly from Lemmas 5 and 7. □

Lemma 9. *For any message M, and any thresholds t, k and n, if $\phi_1 \dots \phi_n = $ column-RSEnc$(M, n-2t, k, n)$, then $\hat{H} = \text{VCom}([(\text{hash}(\hat{\phi}_1), \dots \text{hash}(\hat{\phi}_n)])$ is a binding commitment to M.*

Proof. The proof follows from the correctness of the erasure coding scheme, and the binding properties of the hash, vector commitment and the deterministic polynomial commitment being used. □

Lemma 10. *The AVID construction in Algorithm 2 satisfies the availability property in Definition 1.*

If k honest replicas have finished the dispersal phase then this means that each replica i has a tuple $(\hat{H}, H_{\pi_i}, \phi_i)$ such that VVerify (hash(Com(pp, ϕ_i, t)), H_{π_i}, $\hat{H}$) is True. If an honest client initiates *avid-retrieve* then it will at least receive those k valid tuples and would be able to reconstruct some message M.

Lemma 11. *The AVID construction in Algorithm 2 satisfies the correctness property in Definition 1.*

Proof. By the termination property (shown in Lemma 3 above), we know that for any message M of size $k \cdot (n-2t)\lambda$, if an honest client initiates *avid-disperse*(M), then every honest replica p_i will eventually *avid-deliver*$(\hat{H}, H_{\pi_i}, \phi_i)$ where here we let $\phi_1 \dots \phi_n =$ *column-RSEnc*$(M, n-2t, k, n)$ and $\hat{H} = \text{VCom}([(\text{hash}(\hat{\phi}_1), \dots \text{hash}(\hat{\phi}_n)])$.

If an honest client initiates *avid-retrieve* with row_{indices}, then every online honest replica p_i will eventually send $(\hat{H}, H_{\pi_i}, \hat{\phi}_i, \pi_i)$ such that VVerify(hash$(\hat{\phi})$, H_{π_i}, $\hat{H}$) returns **True**. Here, π_i includes the evaluation of ϕ_i at each specified index, accompanied by an aggregate proof confirming that all evaluations are consistent with the same polynomial ϕ_i. Specifically, $\pi_i = open_{agg}(\text{pp}, \phi_i, row_{\text{indices}})$.

Notice that no inconsistent polynomial can pass the check VVerify against $\hat{H}$ due to the binding property of the vector commitment, polynomial commitment and hash function. Since at least k honest replicas are guaranteed to be online in *avid-retrieve* and $k \geq t$, then the honest client will eventually have at least k elements in the dictionary with $\hat{H}$ as it's key for each row polynomial. Therefore, the honest client will eventually be able to *decode* for each row and *avid-output rows*. □

Lemma 12. *The AVID construction in Algorithm 2 satisfies the Retrieval Consistency property in Definition 1.*

Proof. Due to lemma 9, and the binding property of the polynomial commitment and vector commitment for each message block M, the only way two retrieving clients could output two different message sub-blocks M_x and M'_x for the same set of indices x, is

if each message had a different vector commitment. Assume two honest clients p_1 and p_2, where p_1 called *avid-output*(M) consistent with a commitment $\hat{H}$, and p_2 called *avid-output*(M') consistent with a commitment $\hat{H}'$. This means that p_1 received at least $k \geq t+1$ fragments consistent with a commitment $\hat{H}$. Similarly, p_2 received at least $k \geq t+1$ fragments consistent with a commitment $\hat{H}'$. However, since $k \geq t+1$ this implies that there exist one or more honest replicas that *avid-deliver* different vector commitments and as such sent different vector commitments one with $\hat{H}$ and another with $\hat{H}'$ with contradicts lemma 7 that states (among other things) that all honest replicas should deliver the same vector commitment $\hat{H}$. □

Lemma 13. *The AVID construction in Algorithm 2 is* robust.

Proof. Assume that an arbitrary honest replica i finished dispersal with a fragment $m_i = \hat{H}, \pi_i, \phi_i$. Due to lemma .8, every other honest replica j also finished with a valid m_j with the same $\hat{H}$. Moreover, due to lemma .4, there exists a unique bivariate polynomial $f(x, y)$ of degree $k-1$ in one dimension and $n-2t-1$ in the other dimension where $\phi_i(.) = f(., i)$ and as such there is exist a unique message M (due to the correctness of the bi-dimensional Reed-Solomon encoding). If a client retrieved $M = \textit{column-RSDec}(\phi_{x_1}, \ldots \phi_{x_k})$ where $\phi_{x_1}, \ldots \phi_{x_k})$ is the set of k polynomials that are consistent with $\hat{H}$ then it follows that $\textit{column-RSEnc}(M) = \phi_1 \ldots \phi_n$. Since the vector commitment, polynomial commitment, and hash functions are all deterministic, then if the client *avid-disperse*(M) then every honest replica i will also *avid-output* $(m_i = \hat{H}, \pi_i, \phi_i)$. □

Lemma 14. *The AVID construction described in Algorithm 2 is* committing.

Proof. The committing property follows from Lemma 9 and Lemma 12. □