Practical and Improved Byzantine Reliable Broadcast and Asynchronous Verifiable Information Dispersal from Hash Functions

Nicolas Alhaddad
nhaddad@bu.edu

Sisi Duan
duansisi@mail.tsinghua.edu.cn

Mayank Varia
varia@bu.edu

Haibin Zhang
haibin@bit.edu.cn

***Abstract*—This paper improves upon two fundamental and closely related primitives in fault-tolerant distributed computing—Byzantine reliable broadcast (BRB) and asynchronous verifiable information dispersal (AVID). We make improvements asymptotically (for our AVID construction), concretely (much lower hidden constants), and practically (having 3 steps, using hash functions only, and avoiding using online error correction on the bulk data).**

The state of the art BRB protocol of Das, Xiang, and Ren (DXR BRB, CCS 2021) uses hash functions only and achieves a communication overhead of $O(nL+kn^2)$, where n, L, and k are the number of replicas, the message length, and the security parameter, respectively. More precisely, DXR BRB incurs a concrete communication of $7nL + 2kn^2$, with a large constant 7 for the bulk data term (i.e., the nL term). Das, Xiang, and Ren asked an open question if it is possible "from a practical point of view to make the hidden constants small." Two other limitations of DXR BRB that authors emphasized are that "higher computation costs due to encoding and decoding of the message" due to applying error correcting codes on bulk data and the fact that "in the presence of malicious nodes, each honest node may have to try decoding f times" due to the use of an online error correcting algorithm. Meanwhile, the state of the art AVID protocols achieve $O(L+kn^2)$ communication assuming trusted setup. Apparently, there is a mismatch between BRB and AVID protocols: another natural open problem is whether it is possible to build a setup-free AVID protocol with $O(L + kn^2)$ communication.

In this work, we answer all these open questions in the affirmative. We first provide a hash-based BRB protocol that improves concretely on DXR BRB, having low constants and avoiding using online error correction on bulk data. Our key insight is to encode the consistency proof, not just the message. Our technique allows disseminating the message and proof together. Then we provide the first setup-free AVID protocol achieving $O(L+kn^2)$ communication. Both our BRB and AVID protocols are practical because they have 3 steps, a multiplicative factor of 3 for the bulk data term, use hash functions only, and they avoid applying online error correction on bulk data.

I. Introduction

Byzantine reliable broadcast (BRB) and asynchronous verifiable information dispersal (AVID) are two fundamentally important primitives in fault-tolerant distributed computing. Both primitives allow one to reliably disseminate data among a set of n replicas even in the presence of Byzantine failures. The difference, roughly speaking, is that replicas in BRB obtain a full copy of the message broadcast, while replicas in AVID may store data fragments instead of a full copy of message during the dispersal protocol—as long as these fragments *altogether* allow reconstructing the original message using the retrieval protocol.

A. Existing BRB and AVID Protocols

Since Bracha's broadcast [9, 10], the research of BRB has primarily focused on reducing the communication complexity of BRB protocols; clearly, the $O(n^2)$ message complexity for Bracha's broadcast has already been optimal. Indeed, a number of BRB protocols have been proposed to save the communication complexity [5, 13, 14, 17, 36]. All these BRB protocols use erasure coding or error correcting codes. Nayak, Ren, Shi, Vaidya, and Xiang [36] propose an erasure-coded BRB protocol achieving a communication cost of $O(Ln+kn^2)$ and relying on trusted setup, where n, L, and k are the number of replicas, the message length, and the security parameter, respectively.

In fact, assuming trusted setup, one can use vector commitments [15] to build a generic BRB framework, called VC-BRB, which also achieves $O(Ln+kn^2)$ communication. VC-BRB can be regarded as a generalization of [14] and has been described in [4].

The state of the art BRB protocol of Das, Xiang, and Ren (DXR BRB) achieves a communication overhead of $O(nL + kn^2)$ [17] by relying on hash functions only. DXR BRB uses online error correction (OEC) [7] for the goal.

The related notion of AVID was originally introduced by Cachin and Tessaro [14]. Specifically, their work contributed two constructions for each of AVID and BRB protocols, with the only difference being that AVID stores erasure-coded fragments, while BRB stores the whole message. Their initial AVID/BRB constructions use a list of n hashes (cross-checksum) [23, 31] to achieve $O(nL + kn^3)$ communication; we refer to these protocols as CT0 AVID and CT0 BRB, respectively. Their improved constructions use a Merkle tree to attain $O(nL+kn^2 \log n)$ communication; we call the AVID protocol and the BRB protocol using Merkle tree CT AVID and CT BRB, respectively.

Hendricks, Ganger, and Reiter [27] use fingerprinted cross-checksum to build the HGR AVID protocol that has $O(L + kn^3)$ communication. Alhaddad, Duan, Varia, and Zhang, ECP-AVID, uses an erasure coding proof (ECP) system to achieve $O(L+kn^2)$ dispersal communication and $O(L+kn)$

retrieval communication, albeit at the expense of requiring trusted setup. The same authors also propose a construction ECP2-AVID that does not rely on trusted setup but has higher retrieval communication complexity ($O(L + kn \log n)$). In fact, assuming trusted setup, one could also build a generic framework for AVID using vector commitments, VC-AVID, which can be viewed as being implied by a series of protocols and implementations [1, 4, 14]. We describe VC-BRB and VC-AVID in Sec. IV.

B. Open Problems for BRB and AVID

We describe three open problems we aim to solve in this paper for BRB and AVID protocols. The first two open problems are directly from DXR, where the authors discussed two limitations of DXR BRB [17, Section 8].

Reducing the concrete expansion factor in BRB and AVID protocols. Asymptotically, DXR BRB achieves $O(nL + kn^2)$ communication without trusted setup. However, the authors of DXR BRB acknowledged in [17, Section 8 Discussion] that: "*Although we mostly discuss asymptotic cost in this paper, it is equally important from a practical point of view to make the hidden constants small. Indeed, this is the case with the primitives we construct in this work.*" As the authors pointed out, DXR BRB has a concrete communication cost of $7nL + 2kn^2 + 2n^2$. In particular, the hidden constant 7 in the nL term is quite large and makes DXR BRB less attractive in practical applications. We refer to this hidden constant as the *expansion factor* throughout this work. In the same paper of ECP-AVID [5], the authors also propose ECP-BRB which achieves a concrete communication of $6nL + 2kn^2$ (using trusted setup), where the expansion factor remains large—6. Meanwhile, NRSVX BRB achieves the same communication complexity using trusted setup [36], albeit with more steps than the above two protocols.

In another example, CT BRB (also an AVID protocol) [14] is deemed in practice as one of the most computationally efficient BRB protocols, as it assumes hash functions and authenticated channels only, being very simple to implement. As we have described, it has $O(nL + kn^2 \log n)$ communication. The original design for CT BRB has a concrete communication of $\mathbf{6}nL + 2kn^2 \log n$ with an expansion factor of 6 for its *bulk data term* (i.e., the nL term). The influential HoneyBadgerBFT paper [35]—considered by many to be the first practical asynchronous BFT protocol—implemented CT BRB (code available [1]). When implementing CT BRB, the authors apparently noticed the expansion factor in CT BRB is large and reduced it from 6 to 3 in their implementation. Following the implementation, the known implementations for CT BRB [1, 2, 3, 20, 21, 26] use an optimized version with an expansion factor of 3.

Inefficiency issue for (online) error correction. Another open problem left by Das, Xiang, and Ren in DXR BRB is related to the inefficiency problem of Reed-Solomon (RS) error correcting codes (ECC) [40] and online error correction (OEC) algorithm [7]. In particular, the authors of DXR BRB discussed in [17, Section 8] that: "*One limitation of using ADD in our RBC is its higher computation costs due to encoding and decoding of the message. Additionally, in the presence of malicious nodes, each honest node may have to try decoding f times. Contrary to this, in the RBC protocol of Cachin and Tessaro, each node needs to run the decoding algorithm only once.*" This limitation stems from the fact that DXR BRB uses Reed-Solomon ECC, instead of erasure codes, to encode the message, while using OEC to decode the message in an online manner. However, DXR BRB only addresses the message with the ECC field size. DXR BRB has no limitation on the field size being used and assumes that each symbol fits in the field. Since each symbol is of size $M/(f+1)$ for a message M, the field can grow impractically large. One easy fix may be to set a small constant size for the field. For a field of size GF(2^a), a message can be split into $\frac{M}{a(f+1)}$ polynomials and each fragment i would then be the concatenation of the evaluation of those polynomials at the same point i. However, such an approach is expensive for ECC. For a single polynomial of degree n, the standard error correcting decoding algorithm has a run time complexity of $O(n \log n)$ [22]. For $\frac{M}{a(f+1)}$ polynomials the ECC has to be repeated $\frac{M}{a(f+1)}$ times. The run time becomes even more expensive when running online error code, because the ECC has to be repeated again f times. This would bring the total run time complexity to $O(\frac{M}{a(f+1)} n^2 \log n) = O(\frac{M}{a} n \log n)$. (Note OEC for a vector of small-sized messages outputs a result only if OEC is successful for each small-sized message.)

In contrast, applying erasure coding to a vector of messages would easily bring down the run time complexity to $O(\frac{M}{a})$. In fact, there exist highly optimized erasure coding schemes that can be applied to a vector of small-sized messages, such as Cauchy Reed-Solomon code [38, 39]. If using these erasure codes, the performance difference between OEC and erasure coding is more significant.

AVID without trusted setup. AVID is different from BRB, as it is possible that each replica eventually delivers an coded fragment rather than the original data. While CT BRB and CT AVID share the same complexity, later works, such as HGR AVID by Hendricks, Ganger, and Reiter [27], and recently ECP-AVID by Alhaddad, Duan, Varia, and Zhang [14], have shown CT AVID can be obtained in a much more communication-efficient manner. In particular, ECP-AVID achieves the communication of $O(L + kn^2)$ by using a trusted setup. The concrete communication for ECP-AVID is $6L + 2kn^2$. Clearly, AVID protocols require reducing the expansion factor as well. Note that we have mentioned VC-AVID using vector commitments can also achieve $O(L + kn^2)$ communication.

For BRB protocols, $O(nL + kn^2)$ communication complexity was first attained with the help of trusted setup in [36] and then later achieved by DXR BRB that does not rely on trusted setup. For AVID protocols, while the state of the art protocols have $O(L + kn^2)$ communication, it is still an open question to ask if one can achieve the same communication without the

need of trusted setup.

C. Our Contributions

This paper resolves the three open problems mentioned above, proposing practical BRB and AVID protocols with low expansion factors and a setup-free AVID protocol achieving $O(L + kn^2)$ communication. The protocols avoid applying inefficient OEC to the bulk data. In particular, we make the following contributions:

- We offer a highly efficient BRB protocol (CC-BRB) that asymptotically matches the state of the art BRB protocols, while having an expansion factor of 3. The protocol has 3 steps, using hash functions only. Notably, CC-BRB uses ECC for hashes (of size nk) and uses much more efficient erasure coding for the bulk data.
- We provide a new AVID protocol (CC-AVID) that matches the communication complexity of the state of the art AVID protocols that need trusted setup. CC-AVID uses hash functions only. Our AVID extends our BRB protocol and inherits the low expansion factor property of our BRB protocol.

Concurrent work. Concurrent to our work, Das, Xiang, and Ren provide a new BRB protocol and an AVID protocol [18] that share the same communication complexity as ours. We call their protocols DXR2022-BRB and DXR2022-AVID, respectively. They studied the protocols from a different perspective: can one design protocols with *balanced cost*, meaning that all replicas send the same asymptotic communication? Their protocols achieved the goal. Both protocols in our paper achieve the goal of balanced cost, too.

However, their protocols have one more step than ours (4 steps for them versus 3 steps for our protocols). More importantly, they do not address the two major open problems left by DXR BRB [17]. First, they have even higher expansion factor than DXR BRB, while our protocols have much smaller expansion factor than DXR BRB. Second, they use ECC and OEC more extensively than DXR BRB, while we avoid applying OEC to the bulk data.

D. Our Core Techniques Explained In a Nutshell

Here we briefly describe our core techniques. We focus on BRB protocols, as we feel that BRB appears a bit better-known, though we emphasize that the techniques apply to AVID protocols analogously.

We first observe that it is difficult to improve concrete communication complexity by tweaking the DXR BRB or ECP-BRB protocols directly. DXR BRB internally uses an asynchronous data dissemination (ADD) protocol that inherently needs $7nL$ communication—$1nL$ for the first broadcast phase, and then $2 \times 3nL$ for the two ADD phases involving two all-to-all communication transmitting coded fragments. Moreover, ECP-BRB relies on a proof system where the proof is of the same size of the erasure-coded fragments, which yields $6nL$ communication.

Hence, instead of directly working on the two existing protocols, we design our own. We begin with CT0 BRB of Cachin and Tessaro [14] (that uses cross-checksum). At first glance this may appear to be a strange starting point, since CT0 BRB achieves $O(nL + kn^3)$ communication complexity—that is not just more expensive than the state of the art BRB protocols ($O(nL + kn^2)$), but also than the well-known and widely used CT BRB described in the same paper by Cachin and Tessaro ($O(nL + kn^2 \log n)$).

CT0 BRB uses a hash-based cross-checksum, applying hashes to each erasure-coded fragments and yielding n hashes. It is straightforward to reduce the expansion factor of CT0 from 6 to 3. Our design based on CT0 follows the classic, three-step design: SEND, ECHO, and READY. In the first step, the sender sends each replica an erasure-coded fragment along with all n hashes. The replicas then compute a hash h for the n hashes. Then replicas agree on h and meanwhile *apply ECC instead of erasure codes to the n hashes*. In particular, replicas use ADD to disperse the n hashes, while carefully using both h and the n hashes (in various places) to ensure consistency of reconstructed data. In this way, the communication overhead of transmitting ECC encoded hashes is only $O(kn^2)$, while the expansion factor for the nL term remains 3. Note we use hash functions in a much more fine-grained manner than all prior constructions to ensure consistency of our BRB protocol.

Note, above, our BRB also naturally solves an inefficiency problem due to using Reed-Solomon ECC to the bulk data. In our BRB protocol, we use erasure coding for bulk data, and only apply ECC for n hashes—which does not bottleneck the protocol.

Extending the technique, we provide the first setup-free AVID protocol (CC-AVID) that achieves $O(L + kn^2)$ communication. For the dispersal protocol, we mainly follow the technique of CC-BRB to reduce expansion factor and minimize the use of OEC. For the retrieval protocol, instead of sending the full list of hashes, we use online error correcting on the hashes to achieve the goal of having $O(L+kn)$ retrieval communication as well as achieve low expansion factor.

II. System Model and Problem Statement

We consider Byzantine reliable broadcast (BRB) and asynchronous verifiable information dispersal (AVID) protocols consisting of n replicas, where f out of them replicas may fail arbitrarily (Byzantine failures). We assume the existence of point-to-point authenticated channels between every pair of replicas. We design protocols in asynchronous environments making no timing assumptions. This paper considers adaptive corruption, where the adversary can choose its set of corrupted replicas at any moment during the execution of the protocol, based on the information it obtained so far. The weaker static corruption requires the adversary is restricted to choose its set of corrupted replicas at the start of the protocol and cannot change this set later on. All protocols we consider assume that f is a constant fraction of n with $f \leq \lfloor \frac{n-1}{3} \rfloor$ (which is optimal). A Byzantine *quorum* is a set of $\lceil \frac{n+f+1}{2} \rceil$ replicas. Without loss of generality, this paper may assume $n = 3f + 1$ and a quorum size of $2f + 1$.

Byzantine reliable broadcast (BRB). We review the definition of Byzantine reliable broadcast (BRB). A BRB protocol

BRB protocols	setup	asymptotic communication	concrete communication (omitting insignificant terms)
Bracha's BRB [10]	none	$O(n^2L)$	$\mathbf{2}n^2L$
CT BRB [14]	none	$O(nL + kn^2 \log n)$	$\mathbf{6}nL + 2kn^2 \log n$
DXR BRB [17]	none	$O(nL + kn^2)$	$\mathbf{7}nL + 2kn^2$
ECP-BRB [5]	trusted	$O(nL + kn^2)$	$\mathbf{6}nL + 2kn^2$
VC-BRB ([4, 14]; Sec. IV)	trusted	$O(nL + kn^2)$	$\mathbf{3}nL + 2kn^2$
VC2-BRB ([4, 14]; Sec. IV)	none	$O(nL + kn^2 \log n)$	$\mathbf{3}nL + 2kn^2 \log n$
CC-BRB (this paper)	none	$O(nL + kn^2)$	$\mathbf{3}nL + 9kn^2$

TABLE I: Comparison of BRB constructions. L is the input length and k is the security parameter. A trusted setup means that a trusted dealer is needed to generate public parameters for the system (e.g., the public key for threshold signatures) or replicas have to interactively generate public parameters. VC-BRB is a generic BRB framework using vector commitment (see Sec. IV), while VC2-BRB is an instantiation from the generic framework using Merkle tree. CC-BRB is our core protocol we design in this paper and relies on hash functions (and authenticated channels).

AVID protocols	setup	dispersal communication	concrete dispersal	retrieval communication	concrete retrieval
CT AVID [14]	none	$O(nL + kn^2 \log n)$	$\mathbf{6}nL + 2kn^2 \log n$	$O(L + kn \log n)$	$\mathbf{3}L + kn \log n$
HGR AVID [27]	none	$O(L + kn^3)$	$\mathbf{3}L + 2kn^3$	$O(L + kn^3)$	$\mathbf{3}L + kn^2$
ECP-AVID [5]	trusted	$O(L + kn^2)$	$\mathbf{6}L + 2kn^2$	$O(L + kn)$	$\mathbf{6}L + 2kn$
ECP2-AVID [5]	none	$O(L + kn^2)$	$\mathbf{6}L + 2kn^2$	$O(L + kn \log n)$	$\mathbf{6}L + kn \log n$
VC-AVID ([1, 4, 14]; Sec. IV)	trusted	$O(L + kn^2)$	$\mathbf{3}L + 2kn^2$	$O(L + kn)$	$\mathbf{3}L + 2kn$
VC2-AVID ([1, 4, 14]; Sec. IV)	none	$O(L + kn^2 \log n)$	$\mathbf{3}L + kn^2 \log n$	$O(L + kn \log n)$	$\mathbf{3}L + kn \log n$
CC-AVID (this paper)	none	$O(L + kn^2)$	$\mathbf{3}L + 9kn^2$	$O(L + kn)$	$\mathbf{3}L + 4kn$

TABLE II: Comparison of AVID constructions. Notably, CC-AVID achieves the same communication complexity as ECP-AVID but does not use trusted setup. When describing concrete communication, we omit obviously insignificant terms (e.g., the kn item, the $n \log n$ item). Also, while theoretically one could retrieve data from a set of $2f + 1$ replicas for all approach, we follow the traditional metric and ask all replicas to send fragments during the retrieval protocol.

is specified by two protocols *r-broadcast* and *r-deliver* such that the following properties hold:

- **Validity**: If a correct replica p *r-broadcasts* a message m, then p eventually *r-delivers* m.
- **Agreement**: If some correct replica *r-delivers* a message m, then every correct replica eventually *r-delivers* m.
  previously broadcast by replica p_s.

We remark that as explained in, e.g., [11], agreement implies the properties of consistency and totality.

Asynchronous verifiable information dispersal (AVID). AVID is introduced by Cachin and Tessaro [14] and used to disperse a data block among a set of replicas.

An AVID scheme consists of a dispersal protocol and a retrieval protocol. The dispersal protocol is specified by *avid-disperse* and *avid-deliver*. A client (which may also be a replica) starts *avid-disperse* (id, M), and replicas complete the dispersal protocol and *avid-deliver* M for id. The retrieval protocol is defined by *avid-retrieve* and *avid-output*. In the retrieval protocol, a client ct may trigger *avid-retrieve* and eventually *avid-output* the full block M.

An AVID scheme with tag id should satisfy the following properties with overwhelming probability:

- **Termination**: If *avid-disperse* for id is initiated by a correct client, then *avid-deliver* for id is eventually completed by all correct replicas.
- **Agreement**: If a correct replica completes *avid-deliver* for id, then all correct replicas eventually complete *avid-deliver* for id.
- **Availability**: If a correct replicas completes *avid-disperse* for id, then any correct client that initiates *avid-retrieve* for id eventually reconstructs some block M.
- **Correctness**: If a correct replica completes *avid-deliver* for id, then all correct clients that initiate *avid-retrieve* for id eventually retrieve the same block M. Moreover, If a correct client initiated *avid-disperse*(id, M') then $M = M'$.

Expansion factor. The communication cost of a BRB or an AVID protocol can be (roughly) divided into a bulk data term which involves bulk data and n (e.g., $3nL$, $6nL$ and $7nL$ for BRB protocols, $3L$ and $6L$ for AVID protocols), additional security parameter terms (e.g., $2kn^2 \log n$, $2kn^2$), and optionally some insignificant lower-order terms (e.g., n^2).

The security parameter terms may become significant only when n becomes large, or the bulk data is small itself. But in general, the bulk data term is more practically important, because the bulk data terms are usually bottlenecks for major applications of BRB or AVID protocols. We define the constant in the bulk data term as being the *expansion factor*. Reducing the expansion factor is a practically important research problem as emphasized by Das, Xiang, and Ren in their state of the art BRB protocol [17], and explored by a line of practical BRB and AVID implementations [1, 2, 26, 35].

For example, CT BRB was designed to have an expansion factor of 6, but follow-up implementations [1, 2, 3, 20, 21, 26]

on CT BRB all uses an optimized version that has expansion factor 3. From a different angle, asynchronous BFT protocols, such as HoneyBadgerBFT [35] (CT BRB vs. Bracha's broadcast), BEAT [19] (CT BRB vs. HGR AVID), and WaterBear (CT BRB vs. Bracha's broadcast) [21], have shown the bulk data term is the bottleneck for the throughput of all these protocols, because the other terms, including the security parameter terms, are roughly the same for these BRB or AVID protocols.

III. Building Blocks

State of the art BRB protocols use erasure codes or error correcting codes. Both can encode the data block into fragments. Erasure codes tolerate erasures (unavailable fragments), while error correction codes tolerates errors (incorrect/corrupt fragments). In general, error correcting codes have more restricted syntax and more expensive operations. Looking ahead to our own constructions, we will use erasure codes when possible and error correcting codes only when needed.

Erasure coding scheme. An (m, n) erasure coding scheme over an alphabet $\mathcal{M}$ is a pair of algorithms $(encode, decode)$, where $\mathcal{M}$ denotes the alphabet for a single fragment, *encode* : $\mathcal{M}^m \rightarrow \mathcal{M}^n$ and *decode* : $\mathcal{M}^m \rightarrow \mathcal{M}^m$. The *encode* algorithm takes as input a data block, consisting of m data fragments, and outputs $n > m$ coded fragments. The *decode* algorithm takes as input any m-size subset of coded fragments and outputs the original data block containing m data fragments. Namely, if $[d_1, \ldots, d_n] \leftarrow$ *encode*(M), then *decode*$(d_{i_1}, \ldots, d_{i_m}) = M$ for any distinct $i_1, \ldots, i_m \in [1..n]$. An (m, n) erasure coding scheme is *linear* if each coded fragment d_i $(i \in [1..n])$ is a linear combination of the first m data fragments, *i.e.*, $d_i = \sum_{j=1}^{m} b_{ij} d_j$, where b_{ij}'s are coding coefficients. The coding coefficients for a *generator matrix* for the linear code. An (n, m) erasure coding scheme is *systematic*, if the first m coded fragments are the original m data fragments.

Error correcting code (ECC). An error correcting code allows a sender to add redundancy to the data transmitted such that the receiver can detect and correct a limited number of errors. In this paper we will use the standard Reed-Solomon (RS) error correcting code (ECC) [40]. An (m, n) Reed-Solomon ECC over a finite Galois Field $\mathcal{F} = \text{GF}(2^a)$ has two algorithms *RSEncode* and *RSDecode*. *RSEncode*(M, m, n) takes a message M made of m elements in $\mathcal{F}$ i.e. $|M| = ma$ and produces n fragments in $\mathcal{F}$. In more details, the message M is split into m coefficients of a polynomial P in $\mathcal{F}$. The polynomial of degree $deg = m - 1$ is then evaluated at n different points where $n \geq m$ to produce n fragments, i.e., $fragments = [P(1), P(2), \ldots, P(n)]$. Note that any m fragments can interpolate the polynomial P and reconstruct the message M out of the coefficients of P. *RSDecode*(n', e, deg) takes as input n' fragments, e the number of errors in n' and deg the degree of the polynomial such that $n' \geq 2e + deg + 1$. The algorithm outputs a unique polynomial of degree deg that passes through at least $n' - e$ points in n', or outputs $\perp$ if it cannot find any such polynomial. We leave the details of implementing *RSDecode* to Gao [22]. Note that for efficiency reasons it is better to work in smaller fields (where a is small). For a message M with $|M|$ denoting the number of bits of M, if $|M| > ma$, then m is broken into $M/(ma)$ polynomials, each of which has degree $m - 1$. A fragment i is then the set of all polynomial evaluation at i, i.e., $d_i = f_1(i) \ldots f_n(i)$.

Online error correcting (OEC) algorithm. Online error correcting was first described in [7] and was recently used by [17] to build the state of the art BRB algorithm. OEC uses RS ECC to enable a receiver p_i to reconstruct a message M from n different replicas (where f of these replicas may be faulty), each of which sends a different fragment of M. The OEC algorithm is said to be *online*, because replica p_i receives up to n different fragments in no particular order and can decide when the fragments it received are enough to stop listening for new messages.

Lemma 1 ([7]). *For $n' \geq 2f + 1$ messages received with $e = n' - (2f + 1)$, replica p_i will find $P =$ RSDecode$(n', e, f + 1)$ encoding M. If $n = 3f + 1$ then it is guaranteed that p_i will eventually reconstruct M in an asynchronous system.*

In more details, the algorithm has at most $f + 1$ iterations. Each iteration requires a different pair of values for n' (the number of fragments received so far) and e (the number of errors), while the degree of the polynomial returned is fixed at degree f. The algorithm starts when $n' = 2f + 1$ and $e = 0$. OEC runs *RSDecode* with 0 error(s), i.e., *RSDecode*$(n', 0, f)$ (assuming all $2f + 1$ senders sent correct fragments). If *RSDecode* returns $\perp$, then replica p_i has to wait for another fragment (increasing n' by 1) and increases the number of errors by 1. Otherwise *RSDecode* must have returned a polynomial that agree with all $2f + 1$ points in n' and hence can recover M and the algorithm stops. It is easy to see that the algorithm is guaranteed to stop if $n' = 3f + 1$ and $e = f$ is reached.

Vector commitments. Vector commitments allow a prover to commit to an ordered list of values in such a way that they can later open the commitment at a specific position. The prover can convince a verifier that the opening is correct by generating a proof that the verifier can validate. We review the definition of deterministic vector commitments [15]. A static *vector commitment scheme* $\mathcal{V} = (\text{VSetup}, \text{VCom}, \text{VGen}, \text{VVerify})$ comprises four algorithms that operate as follows:

- $\text{VSetup}(1^k, U, n) \rightarrow \overline{\text{pp}}$ is given a security parameter k, a set U, and a maximum vector length n. It generates public parameters $\overline{\text{pp}}$.
- $\text{VCom}(\overline{\text{pp}}, \vec{v}) \rightarrow c$ is given a vector $\vec{v} \in U^\ell$ where $\ell \leq n$. It outputs a commitment string c.
- $\text{VGen}(\overline{\text{pp}}, \vec{v}, i) \rightarrow w_i$ is given a vector $\vec{v}$ and an index i. It outputs a witness string w_i.
- $\text{VVerify}(\overline{\text{pp}}, c, u, i, w) \rightarrow b$ takes as input a vector commitment c, an element $u \in U$, an index i, and a witness string w_e. It outputs a Boolean value b that should only equal 1 if $u = \vec{v}[i]$ and w is a witness to this fact. Note that this

implies correctness.

Some vector commitments like Merkle trees [34] do not require a trusted setup, while others such as the one by Kate, Zaverucha and Goldberg [29] do. For the later type of vector commitments we will assume that VSetup returns $\perp$. Moreover, we will omit passing $\overline{\mathsf{pp}}$ and assume that all the replicas that use vector commitments have already initialized VSetup correctly.

While some vector commitments have both hiding and binding properties, in this work, we only care about the binding property.

- **Binding.** No polynomial time adversary can compute a vector commitment c, a position i, two elements u and v, and two witnesses w_1 and w_2 such that $\mathrm{VVerify}(c, u, i, w_1) = \mathrm{VVerify}(c, v, i, w_2) = 1$.

Hash. We use a collision-resistant hash function $hash$ mapping a message of arbitrary length to a fixed-length output.

IV. Review of BRB and AVID Protocols

A. CT AVID and DXR BRB

We first describe CT AVID and DXR BRB that are technically more relevant to our protocols. We begin with the two BRB protocols of Cachin and Tessaro [14]: CT0 BRB based on cross-checksum [23, 31] and CT BRB based on Merkle tree. Both protocols follow the three-step communication pattern of Bracha's broadcast.

In CT0, the sender uses an $(f+1, n)$ erasure coding scheme to form n erasure-coded fragments (each being of size $\frac{L}{f+1}$). It also computes the hash of each fragment and forms a cross-checksum with n hashes. In the SEND phase, the sender sends each replica a fragment and the cross-checksum. In the ECHO phase, each replica echoes the its fragment and cross-checksum to all replicas. When receiving $2f+1$ fragments and a matching cross-checksum, a replica first decodes the original block, re-encodes the block to generate all n fragments, computes the hashes of n fragments, and verifies if all hashes match the hashes in the cross-checksum. In the READY stage, each replica broadcasts the cross-checksum and its fragment to all replicas. If a correct replica receives $f+1$ READY messages, it broadcasts the cross-checksum and its fragment to all replicas. In CT0, the communication is upper bounded by both ECHO and READY phases. The concrete communication is:

$$\underbrace{n(\frac{L}{f+1}+kn)}_{\text{the SEND phase}}+\underbrace{n^2(\frac{L}{f+1}+kn)}_{\text{the ECHO phase}}+\underbrace{n^2(\frac{L}{f+1}+kn)}_{\text{the READY phase}},$$

which is about $6nL+2kn^3$ (if assuming optimal resilience of $n=3f+1$ and omitting insignificant terms such as $3L$).

The other BRB protocol by Cachin and Tessaro, CT BRB, uses Merkle tree instead of cross-checksum. In CT BRB, each message is sent with $O(\log n)$ hashes instead of n hashes. The concrete complexity is $6nL + 2kn^2 \log n$.

The state of the art BRB protocol without using trusted setup, DXR BRB, uses asynchronous data dissemination (ADD) to achieve $nL + kn^2$ communication [17]. DXR BRB has provided two BRB protocols, one having 5 steps and the other having 3 steps. We only describe the 3-step DXR BRB that follows the message pattern of Bracha's broadcast. In particular, the first phase broadcasts the whole data m. This strategy, to the best of our knowledge, is first used in Haven [6]. The second phase echoes individual coded fragments and a hash of the whole data h. When receiving $2f+1$ matching ECHO messages with the same coded fragment and h, replicas send READY messages with h. If receiving $f+1$ READY messages with the same h, replicas wait for $f+1$ matching ECHO messages with the same h and send READY messages. Each replica stores all fragments received in the READY messages and uses OEC to obtain a message. If the message matches h, then the replica delivers the message. DXR BRB needs $7nL$ communication—$1nL$ for the first phase and $2 \times 3nL$ for the following two phases.

To study the expansion factor problem in a systematic manner, we consider a BRB framework (VC-BRB) and an AVID framework (VC-AVID) below, both of which use vector commitments. VC-BRB slightly modifies CT BRB and has been described in [4]. VC-AVID can be viewed as a variant implied by [1, 4, 14]. Specifically, the difference between VC-AVID and the implementation in HoneyBadgerBFT [1] is that our VC-AVID checks inconsistencies in the ECHO step, while the latter does it in the READY step. All protocols derived from the frameworks have low expansion factor.

B. VC-BRB and VC-AVID

We recall VC-BRB algorithm in Appendix A. Just as in Bracha's broadcast and CT BRB, VC-BRB also follows the three-step message pattern. In all phases replicas transmit fragments instead of the original data, replicas limit the fragment transmission in the ECHO phase but not in the READY phase (which sends cryptographic values only), and replicas identify inconsistent shares in the ECHO phase using vector commitments.

VC-BRB can encompass the two BRB algorithms by Cachin and Tessaro, because cross-checksum and Merkle tree can be viewed as vector commitment schemes. If assuming trusted setup, one has vector commitment schemes with a constant-size commitment and a constant-size proof [15]. Thus, assuming trusted setup, one obtain a BRB construction that has $O(Ln + kn^2)$ communication and low expansion factor of 3. All the above mentioned instantiations have an expansion factor of 3, because all of them involve all-to-all fragment broadcast in the ECHO message (incurring $n^2L/(f+1) \approx 3nL$ communication). In contrast, all other protocols that have the same communication complexity either have high expansion factor (e.g., DXR BRB [17], ECP-BRB [5]) or have more steps than this one (e.g., [36]).

We present VC-AVID in Appendix A. VC-AVID is slightly different from VC-BRB. As in VC-BRB, assuming trusted setup, we directly obtain an AVID construction that has $O(L+kn^2)$ communication and low expansion factor of 3. Such a scheme matches the asymptotic communication complexity for

the ECP-AVID protocol, but outperforms ECP-AVID that has an expansion factor of 6. One could also use Merkle tree to instantiate an AVID (called VC2-AVID), though this results in a higher communication.

The frameworks allow us to better understand the low expansion factor problem and motivate us to design more efficient schemes. Indeed, for both BRB and AVID protocols, the instantiations using the frameworks match the state of the art BRB and AVID protocols, but they rely on trusted setup and computationally expensive (because of the usage of vector commitments). This paper aims at building practical protocols with no trusted setup.

V. Practical and Improved BRB and AVID Protocols Using Hash Functions

A. CC-BRB

This section first provides a BRB protocol, CC-BRB, with an asymptotic communication complexity of $O(Ln+kn^2)$ and a concrete communication complexity of $\mathbf{3}nL + 9kn^2$. CC-BRB has the same complexity as DXR BRB [17], but differs from it in the expansion factor: CC-BRB has an expansion factor of 3, while DXR BRB has an expansion factor of 7. Meanwhile, CC-BRB avoids using OEC on the bulk data, removing a practical efficiency bottleneck of DXR BRB.

The BRB construction requires only 3 steps. It shares the structure of Bracha's broadcast and "combines" both approaches of DXR BRB [17] and CT BRB [14]. On a high level, the protocol uses the cross checksum of CT BRB [14] to send fragments of a message m (SEND phase and ECHO phase) but use the DXR BRB [17] approach to send fragments of the cross checksum itself (in the ECHO and READY phases). This approach will:

- avoid sending the whole message M in the SEND phase to every replica (like what DXR BRB [17] does), otherwise we would not obtain low expansion factor, and
- avoid sending the whole cross checksum in the ECHO phase (like what CT BRB [14] does), otherwise we would obtain higher than kn^2 communication.

To make the protocol "work," however, we need a very fine-grained and careful treatment for various hashes used. Equally important, we use erasure coding to deal with bulk data, and use inefficient OEC only for n hashes. The minimized use of OEC makes DXR BRB practical for large-size messages.

We describe the pseudocode of CC-BRB in Algorithm 1. Our BRB protocol, CC-BRB, uses both standard erasure coding (*encode*, *decode*) and Reed-Solomon ECC (*RSEncode*, *RSDecode*). We also define two hash functions: $hash\colon \{0,1\}^* \to \{0,1\}^k$ (applied to fragments) and $H\colon \{0,1\}^{nk} \to \{0,1\}^k$ (applied to cross-checksum).

We initialize CC-BRB by creating two empty dictionaries called $fragments_{data}$ and $fragments_{hashes}$ for each replica p_i. Here, $fragments_{data}$ maps each id tag and c to possible data fragments for some message M. Additionally, $fragments_{hashes}$ maps each id tag message of a message M and c to possible fragments of the list of hashes of all n fragment of that message M. Then, CC-BRB proceeds as follows.

- SEND phase: The sender p_s encodes the messages M into n fragments using an erasure code $(f+1, n)$. Each fragment is of size $\frac{L}{f+1}$. The dealer then hashes each fragment and create D, a list of n hashes each of size k. The dealer then sends each replica p_j a SEND message containing the fragment d_j and the list of hashes D.
- ECHO phase: Upon receiving a SEND message, each replica p_i verifies the fragment d_i by checking that $hash(d_i)$ is equal to the i^{th} hash in the cross checksum D. If the check succeeds then: replica p_i uses ECC to encode D into n fragments (each of size k) and stores them in the list $\boldsymbol{\pi}$. Then p_i sends an ECHO message containing the data fragment d_i, fragment π_j (the j^{th} fragment of $\boldsymbol{\pi}$), and $c = hash(D)$ to every replica p_j.
- READY phase: Each replica stores the fragments it received in the ECHO messages. A replica p_i broadcasts a READY message containing c and π_i in two cases:
  1) Replica p_i receives $2f+1$ ECHO messages with the same c and π_i.
  2) Replica p_i receives $f+1$ READY messages with the same c and has not sent a READY message. In this case, p_i waits for $f+1$ ECHO with the same c and π_i and then sends a READY message.

Upon receiving $n-f$ READY messages with the same c, each replica starts decoding. In particular, p_i first decodes the coded fragments of hashes using the *RSDecode* function and outputs D'. It then compares $H(D')$ with c, the value it receives from $2f+1$ READY messages. If $H(D') = c$, p_i waits for at least $f+1$ ECHO messages such that for each coded fragment d_j included in an ECHO message, $hash(d_j) \in D'$. Then p_i decodes the fragments and outputs M. Finally, p_i further encodes M and calculates the list of hashes for the coded fragments. If the list of hashes are consistent with the hashes of the coded fragments, p_i *r-delivers* M. Otherwise, p_i *r-delivers* $\perp$.

Communication complexity. In the following analysis we will assume optimal resilience with $n = 3f+1$. The protocol consists of three steps:

1) SEND: p_s sends one SEND message to all n replicas. Each SEND message consists of the cross-checksum D and the fragment d_i each of size nk and $\frac{L}{f+1}$ respectively. Thus, the total communication complexity for the SEND is $3L + kn^2$.
2) ECHO: Every correct replica p_i sends one ECHO message to all n replicas. Each ECHO message consists of the cross checksum fragment π_j, the data fragment d_j and a hash c where the size of the terms costs $\frac{nk}{f+1}$, $\frac{L}{f+1}$ and k respectively. Thus the total communication complexity for the ECHO is $3nL + 4kn^2$.
3) READY: Every correct replica p_i sends one READY message to all n replicas. Each READY message consists of the cross-checksum fragment π_i and the hash c

Algorithm 1 CC-BRB using hash functions with identifier id and sender p_s. Code shown for replica p_i.

Initialization
  $fragments_{data} \leftarrow \perp$ {dictionary $(id, c) \mapsto$ list of fragments d_j}
  $fragments_{hashes} \leftarrow \perp$ {dictionary $(id, c) \mapsto$ list of fragments π_j}
  $e \leftarrow 0$ {number of errors to be corrected by the online error code}
upon *r-broadcast*(id, M) **and** replica is p_s {step 1: SEND }
  $\boldsymbol{d} \leftarrow$ *encode*(M), $D \leftarrow [hash(d_1), \ldots, hash(d_n)]$
  for $1 \leq j \leq n$, send $(id$,SEND$, d_j, D)$ to p_j
upon receiving $(id$,SEND$, D, d_i)$ from p_s for first time {step 2: ECHO }
  if $hash(d_i) = D_i$
    $c \leftarrow H(D)$, $\boldsymbol{\pi} \leftarrow$ *RSEncode*(D)
    send $(id$,ECHO$, (d_i, \pi_j, c))$ to p_j
upon receiving $(id$,ECHO$, (d_j, \pi_i, c))$ from p_j for first time {step 3: READY }
  $fragments_{data}[(id, c)] \leftarrow fragments[(id, c)] \cup [d_j]$
  if (not yet sent a READY message **and** received $2f + 1$ ECHO messages with the same id, c and same π_i)
    send $(id$,READY$, c, \pi_i)$ to all replicas
upon receiving $(id$,READY$, c, \pi_j)$ from p_j for the first time {verification}
  $fragments_{hashes}[(id, c)] \leftarrow fragments_{hashes}[(id, c)] \cup [\pi_j]$
  if (not yet sent $(id$,READY$, c)$ **and** received $f + 1$ READY messages with the same c)
    wait for $f + 1$ ECHO messages with the same c and π_i
      send $(id$,READY$, c, \pi_i)$ to all replicas
  if $fragments_{hashes}[(id, c)] \geq 2f + 1$ {online error correcting code to reconstruct D}
    $D' \leftarrow RSDec(f + 1, e, fragment_{hashes}[id, c])$
    if $H(D') = c$
      wait for f+1 ECHO message where $hash(d_j) \in D'$ **and** filter $fragments_{data}[(id, c)]$ accordingly
        $M \leftarrow decode(fragments_{data}[(id, c)])$ {reconstruct M from the fragments that are contained in D'}
        $\boldsymbol{d'} \leftarrow encode(M)$
        if $D' = hash(d'_1), \ldots, hash(d'_n)$, *r-deliver*$(M)$
        else *r-deliver*$(\perp)$
      else $e \rightarrow e + 1$ {increase the number of errors to align with the online error correcting code procedure}

respectively. Thus the total communication complexity is $4kn^2$.

Hence the total communication complexity is: $3nL + 9kn^2 + 3L$.

Theorem 1. *CC-BRB (Algorithm 1) is a secure BRB protocol.*

Proof. **Validity.** If a correct replica p_s runs *r-broadcast* for id id and a message M then all correct replicas will pass the check $hash(d_j) = D_j$ and have a valid data fragment of M ($d_j =$ *encode*$(M)[j]$) because of the correctness property of the hash function and the encoding algorithm. Hence, every correct replica p_i will echo to replica p_j a valid data fragment d_i of M, its own cross checksum fragment π_j of D, $c = hash(D)$, and id. Therefore, all correct replicas will receive at least $2f + 1$ ECHO messages with a data fragment and the same c, id and consistent cross checksum fragments. Thus, every correct replica p_i will receive at least $f + 1$ valid data fragments of M whose hashes are contained in D. After that, every correct replica p_i will send a READY message with c, id and π_i. Accordingly, every correct replica will eventually receive collectively at least $2f + 1$ READY messages with the same c, id but $2f + 1$ distinct cross checksum fragments of D, and decodes the fragments to D. Even if some READY messages with invalid cross checksum fragments of D were received, the replica can detect this by virtue of the online error correcting code according to Lemma 1 (since the maximum number of faulty READY messages is f). Hence, every correct replica will be able to reconstruct D and by consequence M. M can be decoded because every correct replica has at least $f + 1$ data fragments whose hashes are contained in D and because of the collision resistance of the hash function.

Agreement. Agreement follows immediately from Lemmas 2-4 below. □

Lemma 2. *If a correct replica p_i r-delivers M with id associated with a cross checksum D and a hash c such that $c = hash(D)$, then every correct replica p_i will eventually receive at least $f + 1$* ECHO *messages with the same id, c and π_i. Additionally, each of these $f + 1$* ECHO *messages will contain a distinct data fragment d_j whose hash is in the cross checksum ($hash(d_j) \in D$). Finally, π_i is a valid fragment of D; that is, $\pi_i =$ encode$(D)[i]$.*

Proof. If a correct replica *r-delivers* M with id, then it must have received $2f + 1$ READY messages with the same id and c. Therefore, $f + 1$ of those READY messages must have been sent by correct replicas. Hence, there is at least one correct replica p_i that received $2f + 1$ ECHO messages with the same id, c and π_i. Since f is the total number of faulty replicas then at least $f+1$ replicas received a SEND message from the sender with id containing a fragment such that the hash is contained in the cross checksum. Therefore, every correct replica p_j will eventually receive at least $f+1$ ECHO messages with the same id, c and π_j each containing a data fragment whose hash is in the cross checksum. Finally, correct replicas who generate the hash c for their ECHO messages must have received D in their SEND message (unless p_s has broken collision resistance of the hash function), and therefore they will generate π_i consistent with D. □

Lemma 3. *If a correct replica p_i r-delivers M with id associated with a cross checksum D and a hash c such that $c = hash(D)$, then every correct replica will eventually be able to reconstruct D.*

Proof. As stated above, if p_i *r-delivers* M then with id, then it must have received at least $f + 1$ READY messages with the same c and id from correct replicas (possibly including p_i itself). Therefore, all other f correct replicas will receive at least $f + 1$ READY messages with the same c and id. By Lemma 2, every correct replica p_j will eventually receive $f+1$ ECHO messages with the same c and id and valid π_j. Thus, they will be able to send their own READY messages with valid π_j. As a result, at least $2f+1$ correct replicas will send READY messages with the same c and id as well as coded fragments that are consistent with the encoding of D. By Lemma 1, it follows that every correct replica will eventually reconstruct D. □

Lemma 4. *If a correct replica p_i r-delivers M with id associated with a cross checksum D and a hash c such that $c = hash(D)$, then every correct replica eventually r-delivers M associated with the same id and D.*

Proof. By Lemma 3, every correct replica will receive the same cross checksum D. Hence, every correct replica can determine the validity of D deterministically and reconstruct M accordingly. Then, either p_i will detect that the cross checksum D is a *valid* cross checksum consistent with some message $M \neq \perp$, or will detect that D is an invalid cross checksum and will *r-deliver* $M = \perp$. Either way, all other correct replicas will also detect D to be valid or invalid in the same way, due to the collision resistance of the hash function and the correctness of the encode and decode algorithm and Lemma 2. □

B. CC-AVID

We now present the first hash-based, setup-free CC-AVID that achieves $O(L + kn^2)$ communication. CC-AVID has 3 steps and an expansion factor of 3, while using RS ECC on cross-checksum instead of the bulk data. The pseudocode is provided in Algorithm 2.

Dispersal protocol. The structure of the dispersal algorithm of CC-AVID is almost identical to that of CC-BRB. In fact, the only difference between them can be reduced to the ECHO message of CC-AVID not containing a data fragment. After all, the whole point of AVID is not to store the whole data in one replica. In more details, the dispersal protocol has three phases: SEND, ECHO, and READY. Each replica maintains three local parameters: $fragments_{hashes}$, $fragments$, and e. The $fragments_{hashes}$ dictionary stores the coded fragments for the list of hashes. The $fragments$ is a dictionary for the coded fragments of the bulk data. The e is a counter for the number of errors, used as input in the *RSDecode* function.

- SEND phase: The sender p_s first encodes M using the standard erasure coding scheme and outputs fragments $\boldsymbol{d}$. Then p_s generates D, a list of hashes for $\boldsymbol{d}$. For each p_j, p_s sends a (id,SEND, d_j, D) message, i.e., the fragment for p_j and the entire vector of n hashes.
- ECHO phase: Upon receiving a SEND message, each replica p_i verifies whether the i^{th} hash in D matches its previously received fragment. If so, it stores the fragment in the $fragments$ dictionary. Then p_s sets c to $H(D)$. It also encodes D into n fragments using the *RSEncode* function and stores them in the list $\boldsymbol{\pi}$. Then p_i sends an ECHO message to each p_j containing π_i and c.
- READY phase: Each replica collects ECHO messages. The replica sends a READY message containing π_i and c under two cases:
  - Replica p_i receives $2f+1$ ECHO messages with the same c and π_i.
  - Replica p_i receives $f+1$ matching READY messages and has not previously sent a READY message. In this case, p_i waits for $f + 1$ matching ECHO messages with the same c and π_i and then sends a READY message.

Upon receiving $2f+1$ matching READY messages, p_i executes the *RSDecode* function on the fragments of the hashes and obtains D'. Then p_i compares $H(D')$ with c, If so, p_i executes *RSEncode* function on D' and verify whether D' is consistent with the $\boldsymbol{\pi}$ it receives in the $2f + 1$ READY messages. If so, p_i *avid-delivers*. Otherwise, p_i continues to collect READY messages until it *avid-delivers*.

Retrieval protocol. A client p maintains three parameters: $frag_{data}$, $frag_{hashes}$, and e. $frag_{data}$ and $frag_{hashes}$ are used to store the coded fragments for the bulk data and the list of hashes, respectively. The e is a counter for the number of errors.

If a replica receives a retrieval request from the client p, it sends a RETRIEVE message to p containing $fragments[id]$. Upon receiving a (id,RETRIEVE, d_j, π_j, c) message from p_j, p first stores d_j and π_j locally. If p receives $2f + 1$ RETRIEVE messages, it starts the decoding processes. This process may continue until p successfully *avid-outputs* some value.

The decoding process proceeds as follows. Client p first decodes the $frag_{hashes}$ (i.e., the coded fragments of the hashes) by executing the *RSDecode* function and outputs D. It then compares each hash in D with the coded fragments in $frag_{data}$ (i.e., coded fragments for the bulk data.) If there are at least $f + 1$ coded fragments in $frag_{data}$ such that the hashes match those in D, p decodes the filtered data fragments and outputs a message M. Client p further encodes M again using the standard erasure coding scheme. It also obtains a list of hashes for the fragments D'. If $H(D')$ is the same as c (the value included in at least $2f + 1$ READY messages), p *avid-outputs* M.

Concrete communication complexity analysis. Again, for simplicity, we assume optimal resilience with $n = 3f + 1$. The dispersal protocol of AVID consists of three steps:

1) SEND: Each SEND message consists of the cross checksum D and the fragment d_i each of size nk and $\frac{L}{f+1}$ respectively. Thus, the total communication for the SEND step is $3L + kn^2$.

Algorithm 2 CC-AVID using hash functions with identifier id and sender p_s. Code shown for replica p_i.

Initialization
  $fragments_{hashes} \leftarrow \perp$, $fragments \leftarrow \perp$ {$fragments_{hashes}$ is a dictionary $(id, c) \mapsto [\pi_i]$, $fragments$ is a dictionary dictionary $id \mapsto d_i$}
  $e \leftarrow 0$
▷ **dispersal**
upon *avid-disperse*(id, M) and replica is p_s {step 1: SEND}
  $\boldsymbol{d} \leftarrow$ *encode*(M), $D \leftarrow [hash(d_1), \ldots, hash(d_n)]$
  for $1 \leq j \leq n$, send $(id, \text{SEND}, d_j, D)$ to p_j
upon receiving $(id, \text{SEND}, D, d_i)$ from p_s for first time {step 2: ECHO}
  if $hash(d_i) = D_i$
    $fragments[id] \leftarrow (d_i, \pi_i, c)$, $c \leftarrow H(D)$, $\boldsymbol{\pi} \leftarrow$ *RSEncode*(D)
    send $(id, \text{ECHO}, (\pi_j, c))$ to all j replicas
upon receiving $(id, \text{ECHO}, (\pi_i, c))$ from p_j for first time {step 3: READY}
  if (not yet sent a READY message **and** received $2f + 1$ ECHO messages with the same id, c and π_i)
    send $(id, \text{READY}, c, \pi_i)$ to all replicas
upon receiving $(id, \text{READY}, c, \pi_j)$ from p_j for the first time {verification}
  $fragments_{hashes}[(id, c)].add(\pi_j)$
  if (not yet sent READY message **and** received $f + 1$ READY messages with the same id, c)
    wait for $f + 1$ ECHO messages with the same id, c and π_i
      send $(id, \text{READY}, c, \pi_i)$ to all replicas
  if $fragments_{hashes}[(id, c)].length \geq 2f + 1$
    $D' \leftarrow$ *RSDecode*$(fragments_{hashes}[(id, c)], e, f)$
    if $H(D') = c$
      $\boldsymbol{\pi} \leftarrow$ *RSEncode*(D')
      if $fragments[id] \neq (*, \pi_i, c)$, $fragments[id] \leftarrow (\perp, \perp, c)$
      avid-deliver(id)
    else $e \leftarrow e + 1$
▷ **retrieval**
Initialization
  $frag_{data} \leftarrow \perp$, $frag_{hashes} \leftarrow \perp$, $e \leftarrow 0$ {e is the number of errors to correct}
upon *avid-retrieve*(id, p) {broadcast fragments}
  if $fragments[id] \neq \perp$
    $(d_i, \pi_i, c) \leftarrow fragments[id]$
    send $(id, \text{RETRIEVE}, d_i, \pi_i, c)$ to p
upon receiving $(id, \text{RETRIEVE}, d_j, \pi_j, c)$ from p_j {verify and decode}
  $frag_{data}[id].add(d_j)$, $frag_{hashes}[(id, c)].add(\pi_j)$
  if $frag_{hashes}[(id, c)].length \geq 2f + 1$ {online error correcting code to reconstruct D}
    $D \leftarrow$ *RSDecode*$(frag_{hashes}[(id, c)], e, f)$, $M \leftarrow recoverMessage(frag_{data}[id], D, c)$
    if $M \neq \perp$, *avid-output*(M)
    else $e \leftarrow e + 1$
▷ **library**
function $recoverMessage(fragments, D, c)$
  $filterd_{fragments} \leftarrow \perp$
  if $(H(D) = c)$
    for $1 \leq i \leq fragments.length$
      if $fragments_i = D_i$
        $filterd_{fragments}.add(fragments_i)$
        if $filterd_{fragments}.length = f + 1$
          $M \leftarrow$ *decode*$(filtered_{fragments})$, $\boldsymbol{d} \leftarrow$ *encode*(M), $D' \leftarrow [hash(d_1), \ldots, hash(d_n)]$
          if $H(D') = c$, **return** M

2) ECHO: Every correct replica p_i sends one ECHO message to all n replicas. Each ECHO message consists of the cross-checksum fragment π_j and a hash c with sizes $\frac{nk}{f+1}$ and k respectively. Thus, the total communication for ECHO is $4kn^2$.
3) READY: Every correct replica p_i sends one READY to all n replicas. Each READY message consists of the cross-checksum fragment π_i and the hash c. Thus, the total communication for READY is $4kn^2$.

Hence the concrete total communication for the dispersal is: $3L + 9kn^2$.

The Retrieval protocol of CC-AVID consists of only one step where each replica sends one message to the client. Each message consists of a data fragment d_i, a cross checksum fragment π_j and a hash c of sizes $\frac{L}{f+1}$, $\frac{nk}{f+1}$ and k respectively. Thus, the total concrete communication complexity is $3L + 4nk$.

Theorem 2. *CC-AVID (Algorithm 2) is a secure AVID protocol.*

The proof of this theorem follows similar techniques as Theorem 1. We defer our security analysis to Appendix B.

VI. Related Work

Both BRB and AVID can be directly used in various reliable storage and communication applications. They are also core building blocks for high-level protocols. BRB [10] can enable applications from online payments [16] to Byzantine fault-tolerant state machine replication (BFT) protocols [26, 30, 35]. Meanwhile, AVID can be used to build asynchronous BFT

storage [19] and blockchain applications such as Rollups [28, 37] and sharding [32].

The definitions of reliable broadcast were informally discussed in SIFT [42]. Later, Schneider, Gries, and Schlichting formalize the notion of reliable broadcast and provide a reliable broadcast implementation in the crash failure model [41]. Reliable broadcast is used in the influential ISIS system by Birman and Joseph [8].

Reliable broadcast and various broadcast variants have been extensively studied and readers may see [12, Chapter 3] (Cachin, Guerraoui, and Rodrigues) for a comprehensive summary.

For Byzantine failures, BRB has also been studied in various other settings, such as probabilistic BRB [25] (which allowing properties to be violated with a fixed and (arbitrarily) small probability) and dynamic BRB [24] (which enabling replicas to join or leave the BRB system dynamically).

Also, it is worth mentioning that prior to CT BRB and CT AVID [14], Cachin, Kursawe, Petzold, and Shoup [13] propose a BRB protocol that improves Bracha's BRB in an "optimistic" way. Namely, if faulty replicas are not actively interfering with the protocol, the communication complexity is $O(nL + kn^2)$; but in the worst case, it has $O(n^2(L + k))$ communication, which is no better than Bracha's broadcast.

Lu, Lu, Tang, and Wang introduce asynchronous provable dispersal broadcast (APDB) that uses threshold signature to disperse erasure-coded fragments to replicas and uses it to build Dumbo-MVBA [33]. APDB leverages vector commitments and threshold signatures and achieves linear message and word complexity. It is a primitive different from AVID or BRB.

VII. Conclusion

This paper propose a novel BRB and an AVID protocol that improve upon state of the art protocols. Both protocols are practical, having 3 steps and low expansion factor, and relying on hash functions only and minimizing the usage of inefficient online error correction. Our AVID protocol is additionally the first setup-free AVID protocol that achieves $O(L + kn^2)$ communication.

Acknowledgments

The first and third author's contribution to this material is based upon work supported by the National Science Foundation under Grants No. 1718135, 1739000, 1801564, 1915763, and 1931714, by the DARPA SIEVE program under Agreement No. HR00112020021, and by DARPA and the Naval Information Warfare Center (NIWC) under Contract No. N66001-15-C-4071.

References

[1] HoneyBadgerBFT library. https://github.com/amiller/HoneyBadgerBFT, 2016.

[2] BEAT library. https://github.com/fifiish/beat, 2018.

[3] HoneyBadgerMPC library. https://github.com/initc3/HoneyBadgerMPC, 2019.

[4] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *PODC*, pages 363–373. ACM, 2021.

[5] Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. Succinct erasure coding proof systems. *Cryptology ePrint Archive*, 2021.

[6] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. High-threshold AVSS with optimal communication complexity. In *Financial Cryptography*, Lecture Notes in Computer Science. Springer, 2021.

[7] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *STOC*, pages 52–61. ACM, 1993.

[8] Kenneth P Birman and Thomas A Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems (TOCS)*, 5(1):47–76, 1987.

[9] Gabriel Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *PODC*, pages 154–162. ACM, 1984.

[10] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[11] Christian Cachin. State machine replication with byzantine faults. In *Replication*, pages 169–184. Springer, 2010.

[12] Christian Cachin, Rachid Guerraoui, and Lus Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. 2nd edition, 2011.

[13] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001.

[14] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *SRDS*, pages 191–201. IEEE, 2005.

[15] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *International Workshop on Public Key Cryptography*, pages 55–72. Springer, 2013.

[16] Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, and Athanasios Xygkis. Online payments by merely broadcasting messages. In *DSN*, pages 26–38. IEEE, 2020.

[17] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *CCS*, 2021.

[18] Sourav Das, Zhuolun Xiang, and Ling Ren. Balanced quadratic reliable broadcast and improved asynchronous verifiable information dispersal. Cryptology ePrint Archive, Report 2022/052, 2022. https://ia.cr/2022/052.

[19] Sisi Duan, Michael K Reiter, and Haibin Zhang. Beat: Asynchronous bft made practical. In *CCS*, pages 2028–2041. ACM, 2018.

[20] Sisi Duan and Haibin Zhang. Pace: Fully parallelizable

bft from reproposable byzantine agreement. *Cryptology ePrint Archive*, 2022.

[21] Sisi Duan, Haibin Zhang, and Boxin Zhao. Waterbear: Information-theoretic asynchronous bft made practical. *Cryptology ePrint Archive*, 2022.

[22] Shuhong Gao. *A New Algorithm for Decoding Reed-Solomon Codes*, pages 55–68. Springer US, Boston, MA, 2003.

[23] Li Gong. Securely replicating authentication services. In *[1989] Proceedings. The 9th International Conference on Distributed Computing Systems*, pages 85–91. IEEE, 1989.

[24] Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, and Andrei Tonkikh. Dynamic byzantine reliable broadcast. In *OPODIS*, 2020.

[25] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. Scalable byzantine reliable broadcast. In *DISC*, 2019.

[26] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous bft protocols. In *CCS*, 2020.

[27] James Hendricks, Gregory R. Ganger, and Michael K. Reiter. Verifying distributed erasure-coded data. In *PODC*, 2007.

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

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

[30] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In *PODC*, pages 165–175. ACM, 2021.

[31] Hugo Krawczyk. Distributed fingerprints and secure information dispersal. In *PODC*, pages 207–218, 1993.

[32] Songze Li, Mingchao Yu, Chien-Sheng Yang, A. Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. Polyshard: Coded sharding achieves linearly scaling efficiency and security simultaneously. In *ISIT*.

[33] Y. Lu, Z. Lu, Q. Tang, and G. Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *PODC*, 2020.

[34] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, 1987.

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

[36] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. Improved extension protocols for byzantine broadcast and agreement. In *DISC*, 2020.

[37] Kamilla Nazirkhanova, Joachim Neu, and David Tse. Information dispersal with provable retrievability for rollups. *Cryptology ePrint Archive*, 2021.

[38] James S Plank, Scott Simmerman, and Catherine D Schuman. Jerasure: A library in c/c++ facilitating erasure coding for storage applications-version 1.2. *University of Tennessee, Tech. Rep. CS-08-627*, 23, 2008.

[39] James S Plank and Lihao Xu. Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications. In *NCA*, pages 173–180. IEEE, 2006.

[40] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of The Society for Industrial and Applied Mathematics*, 8:300–304, 1960.

[41] Fred B Schneider, David Gries, and Richard D Schlichting. Fault-tolerant broadcasts. *Science of Computer Programming*, 4(1):1–15, 1984.

[42] John H Wensley, Leslie Lamport, Jack Goldberg, Milton W Green, Karl N Levitt, Po Mo Melliar-Smith, Robert E Shostak, and Charles B Weinstock. Sift: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, 1978.

APPENDIX

A. VC-BRB

VC-BRB algorithm is depicted in Figure 3. VC-BRB works as follows. Each replica first initializes a local parameter $fragments$, which is a dictionary that maps the id tag and vector commitment c to the coded fragments. In the first SEND step, the sender p_s uses erasure coding to generate erasure-coded fragments and also build a vector commitment and witnesses for all fragments. p_s sends p_i individual fragment d_i, the vector commitment c, and a witness π_i proving d_i is committed in the i^{th} position in c. Upon receiving a SEND message, p_i verifies the correctness of the vector commitment and sends an ECHO message containing (d_i, π_i, c) to all replicas. Upon receiving a valid ECHO message with c, p_i stores the fragment received in the *fragments* dictionary. When there are $2f + 1$ fragments for the same c in the dictionary, p_i decodes the original message and re-encodes the message to generate all n fragments. The p_i computes a vector commitment for these n fragments. If the vector commitment matches c, then p_i broadcasts a READY message with c (but not the fragment) to all replicas. Upon receiving $f+1$ READY message, if p_i has not sent a READY message, it also sends a READY message. If receiving $2f+1$ READY messages with the same c, p_i waits for $f + 1$ valid ECHO messages with the same c and decodes the original message M and then *r-delivers* M.

B. VC-AVID

The pseudocode is shown in Figure 4.

Dispersal protocol. The dispersal protocol involves three steps: SEND, ECHO, and READY, similar to that in BRB. Each replica maintains the $fragments$ parameter, a dictionary to store the received fragments.

Algorithm 3 VC-BRB with identifier id and sender p_s. Code shown for replica p_i.

Initialization
  $fragments \leftarrow \perp$ {dictionary $(id, c) \mapsto$ list of verified fragments d_j}
upon *r-broadcast*(id, M) **and** replica is p_s {step 1: SEND}
  $\boldsymbol{d} \leftarrow$ *encode*(M), $c \leftarrow \text{VCom}(\boldsymbol{d})$, $\boldsymbol{\pi} \leftarrow [\text{VGen}(c, i)$ for $i \in n]$
  for $1 \leq j \leq n$, send $(id, \text{SEND}, c, d_j, \pi_j)$ to p_j
upon receiving $(id, \text{SEND}, c, \pi_i, d_i)$ from p_s for first time {step 2: ECHO}
  if $\text{VVerify}(c, d_i, i, \pi_i) = 1$, send $(id, \text{ECHO}, (d_i, \pi_i, c), i)$ to all replicas
upon receiving $(id, \text{ECHO}, (d_j, \pi_j, c), j)$ from p_j for first time {step 3: READY}
  if $\text{VVerify}(c, d_j, j, \pi_j) = 1$
    $fragments[(id, c)] \leftarrow fragments[(id, c)] \cup [d_j]$
  if (not yet sent (id, READY, c) **and** $fragments[(id, c)].length = 2f + 1$) {received $2f + 1$ consistent ECHO messages}
    $\boldsymbol{d'} \leftarrow$ *encode*(*decode*$(fragments[(id, c)])$)
    $c' \leftarrow \text{VCom}(\boldsymbol{d'})$
    if $c = c'$, send (id, READY, c) to all replicas
upon receiving (id, READY, c) from p_j for the first time {verification}
  if (not yet sent (id, READY, c) **and** received $f + 1$ READY messages with the same c)
    send (id, READY, c) to all replicas
  if received $2f + 1$ READY messages with the same c
    wait for $fragments[id, c].length \geq f + 1$
      $M \leftarrow$ *decode*$(fragments[id, c])$, *r-deliver*(M)

Algorithm 4 VC-AVID with identifier id and sender p_s. Code shown for replica p_i.

Initialization
  $fragments \leftarrow \perp$ {dictionary $id \mapsto$ to a verified fragment d_i, it's proof π_i and the commitment vector c }
▷ **dispersal**
upon *avid-disperse*(id, M) and replica is p_s {step 1: SEND}
  $\boldsymbol{d} \leftarrow$ *encode*(M), $c \leftarrow \text{VCom}(\boldsymbol{d})$, $\boldsymbol{\pi} \leftarrow [\text{VGen}(c, i)$ for $i \in n]$
  for $1 \leq j \leq n$, send $(id, \text{SEND}, c, d_j, \pi_j)$ to p_j
upon receiving $(id, \text{SEND}, c, \pi_i, d_i)$ from p_s for first time {step 2: ECHO}
  if $\text{VVerify}(c, d_i, i, \pi_i) = 1$
    $fragments[(id)] \leftarrow (d_i, \pi_i, c)$
    send (id, ECHO, c) to all replicas
upon receiving (id, ECHO, c) from p_j for first time {step 3: READY}
  if (not yet sent (id, READY, c) **and** received $2f + 1$ ECHO messages with both the same id, c
    send (id, READY, c) to all replicas
upon receiving (id, READY, c) from p_j for the first time {verification}
  if (not yet sent (id, READY, c) **and** received $f + 1$ READY messages with the same c)
    send (id, READY, c) to all replicas
  if received $2f + 1$ READY messages with the same c
    $(d_i, \pi_i, c') \leftarrow fragments[id]$
    if $(c' \neq c)$, $fragments[id] \leftarrow (\perp, \perp, c)$
    avid-deliver(id)
▷ **retrieval**
Initialization
  $fragments \leftarrow \perp$ {dictionary $(id, c) \mapsto$ list of verified fragments d_j}
upon *avid-retrieve*(id, p)
  $(d_i, \pi_i, c) \leftarrow fragments[id]$
  send $(id, \text{RETRIEVE}, d_i, \pi_i, c)$ to p
upon receiving $(id, \text{RETRIEVE}, d_j, \pi_j, c)$ from p_j {verify and decode}
  if $\text{VVerify}(c, d_j, j, \pi_j) = 1$, $fragments[(id, c)] \leftarrow d_j$
  if $fragments[(id, c)].length = f + 1)$
    $M \leftarrow$ *decode*$(fragments[(id, c)])$, $\boldsymbol{d'} \leftarrow$ *encode*$(M))$, $c' \leftarrow \text{VCom}(\boldsymbol{d'})$
    if $(c = c')$, *avid-output*(M)
    else *avid-output*$(\perp)$

In the first step, the sender p_s first encodes its input to generate erasure-coded fragments and also build a vector commitment and witnesses for all fragments. Then p_s sends each p_i a SEND message, consisting of individual fragment d_i, the vector commitment c, and a witness π_i. Upon receiving a SEND message, each replica p_i verifies the vector commitment and then broadcasts an ECHO message containing d_i, π_i, and c. If the message is verified, p_i stores the tuple (d_i, π_i, c) in a dictionary $fragments$. Each replica continues to collect the ECHO messages. If it receives $2f + 1$ READY messages, it broadcasts a READY message containing the vector commitment c. The READY step involves an amplification step such that if a replica receives $f+1$ READY messages with matching c, it also broadcasts a READY message. Finally, if a replica receives $2f + 1$ matching READY messages with the same c. It checks whether its received fragments can be verified by c. If so, the replica directly *avid-delivers*. Otherwise, the replica deletes its local fragment and simply stores a tuple $(\perp, \perp, c)$ in the $fragments$ dictionary.

Retrieval protocol. In the retrieval protocol, a client needs to maintain a dictionary $fragments$ that stores the received

fragments. For each replica, upon receiving the *avid-retrieve* request from client p, the replica obtains the tuple (d_i, π_i, c) in the $fragments$ dictionary and sends a RETRIEVE message to p. Upon receiving an $(id$,RETRIEVE$, d_j, \pi_j, c)$ message from p_j with non-empty d_j and π_j, client p verifies whether the correctness of vector commitment c. If so, p stores the fragment d_j. Upon receiving $f + 1$ valid fragments, p first decodes the fragments to generate M. Then p encodes M again and build a vector commitment c'. It then compares c' with c. If $c = c'$, p *avid-outputs* M. Otherwise, p continues to collect RETRIEVE messages, until it successfully *avid-outputs* some value.

In the remainder of this section, we prove each of the four security properties of an AVID protocol.

Termination. Termination follows immediately from Lemma 5 below. The argument is similar to the one used to prove validity of CC-BRB. We show a complete proof below.

Lemma 5. *If a correct replica initiates avid-disperse for id associated with a message M and a cross checksum D, then every correct replica p_i will eventually avid-deliver id and store the associated cross checksum D and a fragment d_i such that $d_i = encode(M)[i]$.*

Proof. If a correct replica p_s initiates *avid-disperse* for id and a message M then all correct replicas will pass the check $hash(d_j) = D_j$ and have a valid data fragment of M ($d_j = encode(M)[j]$) because of the correctness property of the hash function and the encoding algorithm. Hence, every correct replica p_i will send an ECHO message to each replica p_j its cross checksum fragment π_j of D, $c = hash(D)$ and id. Therefore, all correct replicas will receive at least $2f + 1$ ECHO messages with the same c, id and consistent cross checksum fragments. Thus, every correct replica p_i will send a READY message with c, id and π_i. Therefore, every correct replica will eventually receive collectively at least $2f + 1$ READY messages with the same c, id but $2f + 1$ distinct cross checksum fragments of D that reconstruct to D. Even if some bad READY messages were sent with invalid cross checksum fragments of D to a particular correct replica, the replica can detect this by virtue of the online error correcting code according to Lemma 1 (since the maximum number of faulty READY messages is f). Hence, every correct replica will be able to reconstruct D. In the end, every correct replica has a fragment $d_i = encode(M)[i]$ (from the SEND phase) and will be able to *avid-deliver* id. □

Agreement. The proof is similar to the agreement property of CC-BRB. Notice that as previously stated, the only difference between the CC-BRB protocol and CC-AVID is that in the ECHO phase CC-AVID does not send a fragment of the original data. As such, the lemmas below are nearly identical to Lemmas 2-4. We provide rigorous proofs below for completeness.

Lemma 6. *If a correct replica p_i avid-delivers id associated with a cross checksum D and a hash $c = hash(D)$, then every correct replica p_i will eventually receive at least $f + 1$* ECHO *messages with the same id, c and π_i. Additionally, there exist $f + 1$ correct replicas where each replica p_j will have a distinct data fragment d_j whose hash is in the cross checksum ($hash(d_j) \in D$). Finally, π_i is a valid fragment of D; that is, $\pi_i = encode(D)[i]$.*

Proof. If a correct replica *avid-delivers* with id, then it must have received $2f+1$ READY messages with the same id and c, and at least $f + 1$ of those READY messages must have been sent by correct replicas. Hence there is at least one correct replica p_i that received $2f + 1$ ECHO messages with the same id, c and π_i. Since f is the total number of faulty replicas, at least $f + 1$ replicas have received a SEND message from the sender with coded fragments that are consistent with c. Hence every correct replica p_j will eventually receive at least $f + 1$ ECHO messages with the same id, c and π_j. Finally, correct replicas who generate the hash c for their ECHO messages must have received D in their SEND message (unless p_s has broken collision resistance), and therefore they will generate π_i consistent with D. □

Lemma 7. *If a correct replica p_i r-delivers id associated with a cross checksum D and a hash $c = hash(D)$, then every correct replica will eventually be able to reconstruct D and r-deliver id.*

Proof. As stated above, if p_i *r-delivers* id, then it must have received at least $f + 1$ READY messages with the same c and id from correct replicas (possibly including p_i itself). Therefore, all other f correct replicas will receive at least $f+1$ READY messages with the same c and id. By Lemma 6, every correct replica p_j will eventually receive $f + 1$ ECHO messages with the same c and id and valid π_j. Therefore, they will be able to send their own READY message with valid π_j. As a result, collectively the correct replicas will send $2f + 1$ READY messages, and furthermore their READY messages will contain the same c and id as well as coded fragments that are consistent with the encoding of D. By Lemma 1, it follows that every correct replica will eventually be able to reconstruct D. Hence, every correct replica will eventually be able *r-deliver* id that is associated with D. □

Availability. Availability follows immediately from Lemma 8 below.

Lemma 8. *If a correct replica completes avid-disperse of M for id associated with a cross checksum D such that $c = hash(D)$, then any correct client that initiates avid-retrieve for id will reconstruct M that is consistent with D.*

Proof. If a correct replica completes *avid-disperse* M for id with cross checksum D and hash $c = hash(D)$, then by Lemma 5 all correct replicas will eventually *avid-deliver* id and store the associate cross checksum D and a fragment d_i such that $d_i = encode(M)[i]$. Hence any arbitrary correct client p_i that initiate *avid-retrieve* for id will eventually receive at least $2f+1$ RETRIEVE messages from correct replicas with

consistent cross checksum fragments of D and erasure coded fragments of M together with the same c and the same id. Hence, by virtue of Lemma 1 about the online error correcting code and by the collision resistance of the hash function, every correct client can reconstruct the same M. □

Correctness. Correctness follows from combining Lemma 8 with the lemma below.

Lemma 9. *If a correct replica completes avid-deliver for id associated with a cross checksum D such that $c = hash(D)$, then any correct client that initiates avid-retrieve eventually retrieves the same block M associated with a cross checksum D and c.*

Proof. By Lemma 7, every correct replica will eventually reconstruct D. Moreover by Lemma 6, at least $f + 1$ correct replicas will eventually have fragments whose hashes are contained in D. Hence any arbitrary correct replica p_i that initiates *avid-retrieve* for id will eventually receive at least $2f + 1$ RETRIEVE messages from correct replicas containing consistent cross checksum fragments of D and at least $f + 1$ consistent erasure coded fragments of M together with the same c and the same id. As a result, every correct client can reconstruct D by virtue of Lemma 1 about the online error correcting code and by the collision resistance of the hash function. Therefore, every correct client can determine the validity of D deterministically and reconstruct M accordingly. Then, either p_i will detect that the cross checksum D is a *valid* cross checksum consistent with some message $M \neq \perp$, or will detect that D is an invalid cross checksum and will *r-deliver* $M = \perp$. Either way, all other correct clients will also detect D to be valid or invalid in the same way, due to the collision resistance of the hash function and the correctness of the encode and decode algorithm. □