Decentralized Data Archival: New Definitions and Constructions

Elaine Shi[1]
Computer Science Department and Electrical and Computer Engineering,
Carnegie Mellon University, Pittsburgh, PA, USA

Rose Silver
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Changrui Mu
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We initiate the study of a new abstraction called incremental *decentralized data archival* (iDDA). Specifically, imagine that there is an *ever-growing*, massive database such as a blockchain, a comprehensive human knowledge base like Wikipedia, or the Internet archive. We want to build a decentralized archival system for such datasets to ensure long-term robustness and sustainability. We identify several important properties that an iDDA scheme should satisfy. First, to promote heterogeneity and decentralization, we want to encourage even weak nodes with limited space (e.g., users' home computers) to contribute. The minimum space requirement to contribute should be approximately independent of the data size. Second, if a collection of nodes together receive rewards commensurate with contributing a total of m blocks of space, then we want the following reassurances: 1) if m is at least the database size, we should be able to reconstruct the entire dataset; and 2) these nodes should actually be committing roughly m space in aggregate – specifically, when m is much larger than the data size, these nodes cannot store only one copy of the database, and be able to impersonate arbitrarily many pseudonyms and get unbounded rewards.

We propose new definitions that mathematically formalize the aforementioned requirements of an iDDA scheme. We also devise an efficient construction in the random oracle model which satisfies the desired security requirements. Our scheme incurs only $\widetilde{O}(1)$ audit cost, as well as $\widetilde{O}(1)$ update cost for both the publisher and each node, where $\widetilde{O}(\cdot)$ hides polylogarithmic factors. Further, the minimum space provisioning required to contribute is as small as polylogarithmic.

Our construction exposes several interesting technical challenges. Specifically, we show that a straightforward application of the standard hierarchical data structure fails, since both our security definition and the underlying cryptographic primitives we employ lack the desired compositional guarantees. We devise novel techniques to overcome these compositional issues, resulting in a construction with provable security while still retaining efficiency. Finally, our new definitions also make a conceptual contribution, and lay the theoretical groundwork for the study of iDDA. We raise several interesting open problems along this direction.

2012 ACM Subject Classification Security and privacy → Mathematical foundations of cryptography

Keywords and phrases Decentralized Data Archival

Digital Object Identifier 10.4230/LIPIcs.ITCS.2026.116

Related Version *Full Version*: `https://eprint.iacr.org/2025/969` [37]

Funding This work is in part supported by NSF awards 2212746, 2044679, a Packard Fellowship, and a generous gift from the late Nikolai Mushegian. CM is additionally supported by the National Research Foundation, Singapore, under its NRF Fellowship program, award number NRF-NRFF14-2022-0010. RS is additionally supported by the Carnegie Mellon University CyLab Presidential Fellowship.

[1] Authors are listed in random order.

Acknowledgements We thank Ashrujit Ghoshal for helpful discussion on techniques for proving time-space tradeoff. We also thank Hao Chung for helpful discussons on data availability protocols.

1 Introduction

We consider the problem of building a decentralized data archival system for evolving databases, henceforth called incremental Decentralized Data Archival (iDDA). One primary motivation comes from blockchains. Today, running an Ethereum archival node that backs up the historical transaction logs requires 2TB to 12TB of storage, and the space requirement will continue to grow. A key challenge is how to incentivize nodes to archive the historical logs. In particular, consensus participants only need to maintain the up-to-date state (only 100-200GB today) to remain functional. As a result, the consensus rewards alone do not provide sufficient incentive for storing the entire transactional log. Besides blockchains, iDDA schemes can also be used to build a decentralized backup of the Internet archive (e.g., `archive.org`, hundreds of petabytes in size), or an encyclopedia of human knowledge (e.g., Wikipedia, 10+ TB including all history and media).

In an iDDA scheme, each node will store a (carefully chosen) shard of the dataset, and this shard can evolve over time as the database grows. Furthermore, a node can get remunerated for its contribution through periodical audits. Informally speaking, if a node passes the audit, it means that it has not only committed the purported amount of space S, but is also using this space to store actual data and not junk.

Desiderata. A dream iDDA scheme should satisfy the following desiderata:

1. *Permissionless and low barrier to entry.* We want an open (i.e., permissionless) system, where anyone can join and contribute, using the spare disk space they have on their home computers, without requiring special hardware provisioning. Specifically, this means that 1) the entire data size n can be significantly larger than the any node's available space S; and 2) as the database grows, the nodes need not provision new disk space to continue participation. Philosophically, a low barrier to entry encourages more users to contribute, thus leading to increased decentralization, heterogeneity, and robustness.
2. *Approximate best-possible recoverability.* The strongest recoverability guarantee one can hope for is the following: if any subset of nodes can successfully pass the audit and moreover, their total claimed space is sufficient to hold the entire dataset, then it is possible to reconstruct the dataset in full. However, if each node stores some *random* shard of the dataset, then *strict* best-possible recoverability may be too strong to achieve. Therefore, we relax the notion to an approximate version by allowing for ϵ slack, for an arbitrarily small constant $\epsilon \in (0, 1)$. Specifically, we require that if any μ nodes–each claiming to have committed S space–can all pass the audit, and $\mu \cdot S \geq (1+\epsilon)n$, then we can successfully reconstruct the entire dataset.

   The ϵ-best-possible-recoverability notion can also be further generalized to require that if any μ nodes each with purported S space can pass the challenge, then we can extract roughly $\min(n, (1-\epsilon) \cdot \mu \cdot S)$ amount of useful entropy (assuming a randomly chosen database).
3. *Replication security.* To get α times the fair rewards, a node must commit at least $(\alpha - \epsilon) \cdot S$ blocks of space, where $\epsilon \in (0, 1)$ is an arbitrarily small constant. This definition ensures that a powerful node with ample space cannot store only one copy of the database, and be able to impersonate arbitrarily many nodes and request unbounded rewards. Instead, we want to ensure that if a node is obtaining rewards commensurate with αn

copies of the database, it has indeed dedicated roughly αn amount of space. Inspired by prior works [22, 23, 34], our replication security notion implies an ϵ-Nash equilibrium for rational nodes, that is, a node cannot earn noticeably more rewards if it deviates from honest behavior.

The combination of the above requirements necessitates a strong security definition. In particular, due to the permissionless nature, it could be that the μ nodes that can pass the audit (in the approximate best-possible recoverability or the replication security notions) are in fact pseudonyms controlled by the same adversary. To handle this challenge, our formal definitions implicitly require that security must hold even if *all participating storage nodes are adversarially controlled*.

Such an adversary can mount powerful attacks to gain an advantage, and our scheme must defend against such attacks. For example, if each node's shard assignment is dependent on its identity (e.g., public key), then an adversary can choose a set of identities to maximally overlap the blocks assigned to the adversarial identities. This may allow the adversary to claim k times of the fair reward without committing k times the required space. The adversary can also maliciously select identities to censor specific blocks, thus undermining availability.

Why prior works fail to solve iDDA. Although our iDDA abstraction may bear superficial resemblance to known abstractions such as proofs of retrievability (PoR) [4,11,13,21,30,32,35,36,38], data availability sampling (DAS) [3,14,26–28], proofs of replication (PoRep) [22,22,34][2], and verifiable information dispersal (VID) [7,12,24,29,33,41], none of these known abstractions are adequate for solving the iDDA problem. To the best of our knowledge, all prior works make one or more of the following (implicit) assumptions:

- The node is being audited for storing the entire database (or block), and not a piece of the database – an assumption made by prior works on PoR [4, 11, 13, 21, 30, 32, 35, 36, 38], DAS [14, 27, 28], and PoRep [22, 22, 34].
- A separate instance of the scheme is applied on a per-block basis – an assumption made by prior works on DAS [14, 27, 28], and VID [7, 12, 24, 29, 33, 41]. In Section 1.2, we argue why this "separate instance per block" paradigm cannot satisfy our requirements.
- A threshold fraction of the players must be honest – an assumption typically made by the VID line of work [7, 24, 33].
- The database is static. The aforementioned approach of applying a separate instance per-block can also be viewed as having many small, static databases.

We provide a more detailed comparison with the related work in Section 1.2.

1.1 Our Results and Contributions

New definitions. To the best of our knowledge, we are the first to initiate a formal treatment of the iDDA problem. We provide formal definitions that mathematically capture the aforementioned intuitive requirements of approximate best-possible recoverability and replication security. Importantly, as mentioned, our definitions implicitly require that security hold even when the adversary controls all participating pseudonyms. In particular, we stress that "honest majority" style assumptions are not a great fit in a decentralized/permissionless setting when an adversary can generate arbitrarily many pseudonyms.

[2] In this paper, we use PoR to refer to proofs of retrievability, and we use PoRep or "replication encoding" to refer to proofs of replication.

Efficient construction. We construct an efficient iDDA scheme in the random oracle model that supports append-only updates. Our scheme achieves low barrier to entry: the minimal space provisioning required to contribute is only polylogarithmic. Furthermore, the amortized update cost is logarithmic (or slightly more than logarithmic) for each node as well as the publisher. Importantly, we prove the security of our construction even when all the pseudonyms in the system are controlled by the same adversary.

Below, let λ be the security parameter, let B be the number of bits per block, let S be the space provisioning per storage node, let N be the maximum database size, and let n_0 be the initial database size. Note that S, N, and n_0 are measured in the number of blocks.

▶ **Theorem 1** (Main theorem). *Assume the random oracle model. Let $\epsilon \in (0,1)$ be an arbitrarily small constant. Further, suppose $B \geq \mathsf{poly}(\lambda)\log\lambda$ and $S \geq \mathsf{poly}(\lambda)\log\lambda$ for some suitable polylogarithmic function, and $n_0 \geq \max(S,\lambda)$. There exists an* iDDA *scheme that satisfies ϵ-best-possible recoverability as well as ϵ-replication security, with the following performance bounds where the costs are amortized over $N - n_0$ number of updates, and the $\omega(1)$ term represents an arbitrarily small super-constant function in λ:*

- *Amortized per-node download bandwidth: each update incurs $B \cdot O(1 + S\log\log\lambda/N)$ download bandwidth which is simply $O(B)$ for $S = O(N/\log\log\lambda)$;*
- *Amortized per-node computation: each update incurs $O(B\cdot\log N)\cdot\omega(1)$ node computation for some arbitrarily small super-constant function $\omega(1)$.*
- *Publisher computation: the publisher pays $B \cdot e^{O(1)/\epsilon}\cdot\log N$ computation per update to maintain its data structure;*
- *Audit cost: $B\cdot\log\lambda\cdot\log N\cdot\omega(1)$.*

We stress that under our security requirements, it is inevitable that each node must incur at least constant cost per update. Since otherwise, if 1% of the nodes are not aware of the new update, the adversary can erase the other 99% of nodes and cause this new block to be lost, even if the remaining 1% nodes actually have enough space to store the entire dataset. In this sense, our per-update costs are (nearly) optimal.

Our scheme can also be easily extended to a setting where nodes have heterogeneous space provisioning, provided that the minimum space per node S is at least polylogarithmic. In this non-uniform space setting, a node with $k \cdot S$ need not incur k times the update and audit costs – it still enjoys the same update and audit costs as stated above. See Section 7.1 of the full version for more details.

Technical highlight. We first show how to combine techniques from PoR and PoRep in a non-blackbox fashion to get a decentralized data archival scheme for a static database (see Section 3.1 here and Section 5 of the full version). The most naïve way to get a dynamic scheme is to rerun the static scheme upon every update, but the cost per update would be linear in the data size. The key question is how to make the scheme dynamic without this prohibitive cost.

At first sight, it might be tempting to think that directly applying the standard hierarchical data structure of Bentley and Saxe [9] can turn any static scheme into a dynamic one. Recall that the hierarchical data structure divides the dataset into logarithmically many levels, of size $1, 2, 4, \ldots, n$, respectively, where the smaller levels contain the fresher data blocks. It then runs a separate instance of the static scheme per level. Each level ℓ of size 2^ℓ needs updating only every 2^ℓ steps, and thus the amortized update cost is logarithmic.

Unfortunately, we observe this approach does not generically work for any cryptographic scheme. In our case, there are two reasons why a straightforward application of the hierarchical data structure fails:

1. *Our security definition lacks compositional guarantees.* Our notion of ϵ-best-recoverability includes an implicit accounting requirement: data recovery is only possible if the total storage provisioned by all nodes that pass the audit slightly exceeds the size of the dataset. However, when we have multiple instances of the static scheme – each corresponding to one of the logarithmically many levels – this condition is not necessarily met in a synchronized fashion across all levels. As a result, even if each instance individually satisfies ϵ-best-recoverability, their combined system may fail to satisfy the same notion globally.
2. *The underlying cryptography lacks compositional guarantees.* We make use of a PoRep scheme to compute a replication encoding for each level in the hierarchical data structure. To formally prove security, we need the underlying PoRep scheme to satisfy a certain form of *adaptive sequential composability*, that is, we want the PoRep's security to hold even the adversary may choose some instance's data in a way that depends on another instance's replication encoding. Unfortunately, known PoRep constructions [22, 23, 34] do not provide the desired compositional guarantees.

To solve the first challenge, we devise a new space allocation scheme to allocate a node's space among the multiple levels – see Section 3.2 for details. For the second challenge, we are not aware of any approach to extend the proofs in earlier works [22, 23, 34] to satisfy the desired adaptive sequential composition. Specifically, existing techniques for proving space-time tradeoffs through direct incompressibility arguments are highly involved and apply to extremely limited settings [19,20]. While some other proof techniques [1,2,18,25,39] have been shown to prove space-time tradeoffs, they do not produce meaningful results in our setting. Instead, we devise a method to side-step the lack of composition of the underlying PoRep. Specifically, we modify our construction and force a storage node to locally recompute the replication encodings of all levels upon every update, using the new digest of the entire database to seed the underlying random oracle used in the PoRep scheme. We show that with this modification, we can prove security even when the underlying PoRep scheme does not provide the desired compositional guarantees.

Unfortunately, this modification also incurs significant extra costs. Specifically, each storage node would now have to pay at least $B \cdot S$ cost per update to recompute the replication codes of all levels, where B is the block size and S is the blocks of space allocated by a node. In Section 3.4, we devise some additional algorithmic tricks to avoid this extra cost blowup, and bring the amortized update cost back down, to $\widetilde{O}(1) \cdot B$.

Philosophical discussions: separate archival and retrieval services. In our definitions, we adopt the same philosophy suggested in a line of prior works [7, 38]. Specifically, we do not aim to support efficient read (by index) in the iDDA abstraction itself. This is a deliberate choice, as efficient (authenticated) read can easily be handled with a separate (possibly distributed) retrieval service provider who may simply store a cleartext copy of the dataset along with the Merkle openings [7, 38], allowing users request any specific block. A separate reward system can be used to incentivize the retrieval provider. Moreover, the retrieval service can be designed to optimize efficiency without worrying about redundancy and robustness. Philosophically, by decoupling the problem of providing retrieval from that of data archival, this definitional approach expands the design space and allows for more efficient constructions. For example, our construction can leverage the more efficient erasure codes rather than locally decodable codes.

Open questions. Our work raises several natural open questions. First, although we manage to side-step the underlying PoRep's lack of composition, our work nonetheless leaves open the following interesting question: How can we design a PoRep scheme with the desired adaptive sequential composition property? Likely the reason why the prior works [22, 23, 34] never considered this natural compositional notion is exactly because they (implicitly) considered a setting with a static database, where every node has ample space to store the entire database. A related open question is whether we can design an iDDA scheme that is itself composable. Another theoretically interesting question is whether the random oracle needed for the shard selection can be avoided. On the practical front, it would be interesting to devise a practical variant of our construction. Specifically, for the blockchain context, instantiating the publisher without trust using efficient Incrementally Verifiable Computation (IVC) would be highly relevant – see Section 7.2 of the full version for details. Our paper focuses on append-only updates, so a natural direction is to extend the scheme to support other types of updates.

Roadmap. In the following subsection, we will describe related work in more detail. In Section 2, we will define the iDDA problem and corresponding security definitions more formally. In the last section of this article, we will present an informal technical roadmap describing our constructions.

1.2 Related Work

We now explain in more detail why our iDDA abstraction is different in nature from other abstractions that have been studied in the past.

Why "one instance per block" cannot satisfy our requirements. While the prior works on Data Availability Sampling (DAS) [3, 14, 26–28] and and Verifiable Information Dispersal (VID) [7, 12, 24, 29, 33, 41] may bear superficial resemblance to our problem at first sight, these works are of fundamentally different nature, partly because they (implicitly) assume that a separate instance of the scheme will be applied on a per-block basis.

We argue that the "separate instance per block" approach cannot satisfy our requirements. Under this approach, either almost all nodes must store some (encoded) fragments of *every* block, or only a subset of the nodes are responsible for a block. The former case necessitates per-node storage that is linear in the dataset size, thus violating the "low barrier to entry" requirement; whereas the latter case is prone to a selective censorship attack if all nodes responsible for a block become corrupted. Another way to see this is that even if μ nodes can pass the audit and their purported total space exceeds the data size, we still may not be able to recover the dataset, since the identities of these nodes can be adversarially chosen such that none of them is responsible for storing a particular block.

Next, we review the prior works on PoR, DAS, PoRep, and VID one by one, and give more reasons why all of them are of fundamentally different nature from our iDDA abstraction, despite bearing some superficial resemblance at first sight.

Proofs of retrievability and data-availability sampling. In proofs of retrievability (PoR) [4, 11, 13, 21, 30, 32, 35, 36, 38], an untrusted node can prove that it is indeed correctly storing the data it is asked to store, and that no data loss has occurred. The security definition requires that if a node can successfully pass the audit, then we can extract the entire dataset by rewinding the node and feeding it with many different challenges. A couple works have explored how to extend PoR schemes to support a dynamically evolving database [13, 38].

Data availability sampling [10, 14, 17, 27, 28] can be viewed as a strengthening of PoR. Specifically, in PoR, we assume that the committer of the data is trusted, whereas in DAS, the committer can be adversarial. In comparison with PoR, DAS additionally requires that even when the commitment to the data is adversarially chosen, if a node can pass the audit, we must be able to extract some dataset consistent with the commitment by rewinding the node and feeding it with many different challenges.

Due to historical reasons, PoR was studied typically with the cloud setting in mind, where a client outsources a dataset to a powerful but untrusted cloud server capable of storing the entire dataset. By contrast, the DAS abstraction was proposed in a blockchain context, where blocks are proposed by untrusted block producers in the underlying the consensus protocol. Lightweight consensus clients want to ensure that the data block is available, without necessarily downloading the entire block. It is implicitly assumed that a separate DAS instance will be applied to each block being produced. Because block producers are untrusted, it is crucial that security hold even when the committer is malicious.

Clearly, neither PoR nor DAS solve our iDDA problem for two main reasons. First, the approach of spawning a separate instance of the scheme per block inherently requires each storage node to maintain space that is linear in the size of the database. This directly violates our "low barrier to entry" requirement. Second, neither PoR nor DAS provide replications security. Specifically, a node with ample space can store just one copy of the data, and yet pretend to be arbitrarily many pseudonyms and earn unbounded rewards.

Verifiable information dispersal. In verifiable information dispersal (VID) [7, 24, 33], a potentially malicious party encodes some data string and distributes the encoded fragments among a set of nodes. At the end, the nodes can interact to determine whether the original block is available. VID's security relies on a threshold number of honest nodes, making it unsuitable for a permissionless setting in which the adversary can control arbitrarily many pseudonyms. Like DAS, the VID abstraction was also proposed in a blockchain consensus context, where the committer is a possibly malicious block producer. Consequently, it is typically assumed that a separate VID instance is applied to disperse each block, thus leading to a linear space requirement per node.

Proofs of replication. Proofs of replication (PoRep) [22, 23, 34] guarantee that if a node can pass the audit purporting to have allocated $\alpha \cdot n$ amount of space where n is the data size, then the following are guaranteed: 1) the node must indeed be consuming $(\alpha - \epsilon) \cdot n$ amount of space; and 2) the space is indeed used to store useful data. Existing works on PoRep [22, 23, 34] typically consider a static database and assume that the storage node can store the entire database. PoRep (over unencoded data) also does not guarantee extraction of the entire data when the storage provider can pass the audit.

Polynomial commitment. In our paper, we commit to some data string by computing an erasure code and a vector commitment over the erasure code. We use a constant redundancy for the erasure code (dependent on ϵ). This way, the publisher can simply cache all the opening proofs cheaply. This way, when an update occurs or upon first joining, the storage node only needs to download some data from the publisher, and the publisher need not perform any online computation.

Alternatively, we can use a polynomial commitment scheme with arbitrarily large redundancy (e.g., as large as the underlying field size) [8, 31, 40]. Unfortunately, this approach has the drawback that the publisher must compute the opening proofs on the fly, and in a

non-batched setting, the computation cost is at least linear in the database size using known approaches [8, 31, 40]. In comparison, our approach has only $B \cdot \widetilde{O}(1)$ amortized publisher cost per update.

2 Definitions

In this section, we formally define the iDDA problem and corresponding security definition. Throughout the rest of the paper, we use Σ to denote some finite alphabet. We will treat the database $\mathsf{DB} \in \Sigma^n$ as containing n *blocks*, where each block belongs to Σ. We often use the notation $B := \log_2 |\Sigma|$ to denote the block size. We treat B as a global parameter, so we do not carry it around in the definitions below.

2.1 Syntax of the iDDA Problem

Consider an evolving database whose length increases as new blocks get added over time. An incremental Decentralized Data Archival (iDDA) scheme is a suite of algorithms and protocols involving a data publisher, a set of storage nodes, and an auditor. We now describe the syntax below:

- $\mathsf{crs} \leftarrow \textbf{Setup}(1^\lambda)$: a setup algorithm that takes as input the security parameter 1^λ and outputs a common reference string crs.
- $(\phi, \mathsf{st}^0) \leftarrow \textbf{Init}(\mathsf{crs}, S, \mathsf{DB})$: an initialization algorithm that is executed once upfront by the data publisher. The algorithm takes as input the common reference string crs, a parameter $S \in \mathbb{N}$ that denotes the blocks of space provisioned by each storage node, and an initial database $\mathsf{DB} \in \Sigma^{n_0}$ of length n_0. The algorithm outputs a public digest denoted ϕ and updates the data publisher's internal state to st^0.
- $(\mathsf{st}^0, \mathsf{st}^{id}) \leftarrow \textbf{Join}(\mathsf{st}^0, id)$: a protocol between the data publisher whose starting state is st^0 and a storage node with unique identifier id. At the end of the protocol, the storage node receives state st^{id}, and the data publisher's state st^0 is updated.
- $(\phi, \mathsf{st}^0, \{\mathsf{st}^{id}\}_{id \in \mathsf{IDset}}) \leftarrow \textbf{Update}\left((\mathsf{st}^0, \mathsf{upd}), \{\mathsf{st}^{id}\}_{id \in \mathsf{IDset}}\right)$: a protocol between the data publisher whose current state is st^0 and who receives an update upd as input, and a set of storage nodes whose identities form the set IDset and whose current states are $\{\mathsf{st}^{id}\}_{id \in \mathsf{IDset}}$. At the end of the protocol, the data publisher's internal state st^0 and the nodes' internal states $\{\mathsf{st}^{id}\}_{id \in \mathsf{IDset}}$ are updated, and everyone receives an updated public digest ϕ.
- $b \leftarrow \textbf{Audit}((\mathsf{crs}, \phi, id), \mathsf{st}^{id})$: a protocol between a storage node with identity id and state st^{id} and an auditor whose input is (crs, ϕ, id). At the end of the protocol, the auditor outputs either accept or reject, and the storage node outputs nothing, and its internal state is unchanged.

Without loss of generality, we may assume that the security parameter λ is included in crs, in the data publisher's internal state, and in each storage node's internal state. In general, the auditor can be a different entity from the publisher. For example, as mentioned in Section 7.2 of the full version of the paper, in a blockchain context, the auditor is likely the blockchain such that nodes can get rewarded for passing the audit. On the other hand, the publisher is a separate service provider that maintains hash digests of some data structure built over the blockchain data. Section 7.2 of the full version describes how to remove the trust in the publisher using IVC.

In practice, it might be desirable to periodically invoke the **Audit** protocol, and randomize the time at which it is invoked. This can thwart attacks where the adversary restores its state right before the **Audit** and deletes the state again afterwards – see the end of Section 2.3 for more discussions.

Correctness. We now define the correctness property of an iDDA scheme. At a high level, correctness means that even if there are corrupt nodes in the system, honest nodes can always pass the audit with probability 1. More formally, for any λ, S, any initial DB, any adversary $\mathcal{A}$, the following experiment outputs 1 with probability 1:

- **Initialize.** Call $\mathsf{crs} \leftarrow \mathbf{Setup}(1^\lambda)$, and $\phi, \mathsf{st}^0 \leftarrow \mathbf{Init}(\mathsf{crs}, S, \mathsf{DB})$.
- **Queries.** The adversary can adaptively issue the following queries, where each query is one of the following:
  - *Corrupt.* $\mathcal{A}$ declares some identity id as corrupt. If id has already joined earlier, give its private state st^{id} to $\mathcal{A}$.
  - *Join.* $\mathcal{A}$ specifies an identity id. If the identity id has previously been corrupted, the publisher performs the **Join** protocol with $\mathcal{A}$ who acts on behalf of id. Otherwise, spawn an honest identity id, and have the publisher perform the **Join** protocol with id. If id has previously been spawned, simply ignore the existing state and respawn the node.
  - *Update.* $\mathcal{A}$ specifies an updated block upd, a set of identities IDSet. It is required that IDSet is chosen among the identities that have joined, and moreover, all honest identities that have been joined must belong to IDSet. Now, the **Update** protocol is invoked with upd, involving the publisher and IDSet. Note that $\mathcal{A}$ will act on behalf of any corrupt identity in IDSet.
- **Challenge.** $\mathcal{A}$ specifies an identity id that has joined and remains honest. Now, the honest id engages in an **Audit** protocol with the auditor who receives the up-to-date ϕ. The experiment outputs 1 if the auditor accepts; else it outputs 0.

▶ Remark 2 (Additional desirable properties of our construction). While the above definitions aim to be more general, the constructions proposed in this paper enjoy some additional nice properties: 1) our security notions defined below are respected even when the adversary can see the publisher's state st^0; 2) during the **Join** protocol, the newly joining node simply downloads some portion of the publisher's state st^0, and the **Join** protocol does not alter st^0; and 3) during the **Update** protocol, the publisher updates its state st^0 based on the incoming block, and each storage node then downloads some necessary parts of the new st^0.

In particular, these desirable properties make it possible for us to instantiate the publisher without any trust by relying on an IVC scheme, provided that there is a trusted hash digest of the original dataset (e.g., coming from the blockchain's consensus layer) – see Section 7 of the full version for more details.

We now proceed to the security definitions. Our security definition has two components: approximate best-possible recoverability and replication security. We describe both components in detail below.

2.2 Security Definition: Approximate Best-Possible Recoverability

2.2.1 Intuition of the Definition

Below, we will first define a security game denoted $\mathsf{RecvExpt}$ that allows an adversary $\mathcal{A}$ to interact with the iDDA scheme. After obtaining the common reference string crs, the adversary $\mathcal{A}$ is allowed to pick an initial database of its choice. Then, at any point in time,

the adversary can 1) ask the challenger to run the **Join** protocol with any identity of its choice; and 2) ask the challenger to run the **Update** protocol (with all identities that have joined), supplying any new database update of its choice.

At the end of the security game, we enter a challenge phase. During the challenge phase, the adversary specifies μ challenge identities, and the auditor will run the **Audit** protocol with these challenge identities. If all μ identities succeed in passing the audit, we want to extract a portion of the database whose size is roughly commensurate with the total space of all the μ nodes, that is, roughly $(1-\epsilon)\mu S$ blocks of information. To capture this intuition, our definition below involves a compressor algorithm $\mathcal{C}^{\mathcal{A}}$ and an extractor $\mathcal{E}^{\mathcal{A}}$. Intuitively, the compressor's job is to output the missing $n-(1-\epsilon)\mu S$ blocks of information – denoted $\mathsf{DB}^{\text{short}}$ – that $\mathcal{A}$ may not know. Now, upon receiving this missing information $\mathsf{DB}^{\text{short}}$, the extractor $\mathcal{E}^{\mathcal{A}}$ can extract the entire database. Both algorithms are allowed to interact with the oracle $\mathcal{A}$, including rewinding $\mathcal{A}$ and supplying it with fresh randomness on every invocation.

2.2.2 Formal Definition

Formally, given parameters $\lambda, S, \mu \in \mathbb{N}$, define the following experiment $\mathsf{RecvExpt}^{\mathcal{A}}(1^\lambda, S, \mu)$ between an adversary $\mathcal{A}$ and a challenger:

Experiment $\mathsf{RecvExpt}^{\mathcal{A}}(1^\lambda, S, \mu)$:

- **Initialization**. Run $\mathsf{crs} \leftarrow \mathbf{Setup}(1^\lambda)$. The adversary $\mathcal{A}(1^\lambda, S, \mu, \mathsf{crs})$ specifies an initial database $\mathsf{DB} \in \Sigma^{n_0}$ of length n_0, and the challenger runs $\phi, \mathsf{st}^0 \leftarrow \mathbf{Init}(\mathsf{crs}, S, \mathsf{DB})$ and sends ϕ to $\mathcal{A}$.
- **Queries.** The adversary $\mathcal{A}$ can adaptively make **Join** and **Update** queries. In response, the challenger acts on behalf of an honest data publisher and always keeps track of the publisher's latest state. More precisely:
  - **Join**: $\mathcal{A}$ asks the challenger to run the **Join** protocol with itself. During this protocol, $\mathcal{A}$ acts on behalf of the newly joining node, and it can act arbitrarily. This means that $\mathcal{A}$ can also arbitrarily choose the identity of the newly joining node.
  - **Update**: $\mathcal{A}$ chooses some update upd and asks the challenger to run the **Update** protocol with all the identities that have joined so far. The publisher uses its latest state and upd as input. $\mathcal{A}$ acts on behalf of all the storage nodes and can behave arbitrarily during the protocol.
- **Challenge.** At some point, suppose that the database size has increased to $n \geq n_0$ following the **Update** queries. Now, for $j = 1$ to μ sequentially: the adversary $\mathcal{A}$ specifies id_j, and the challenger, acting on behalf of the auditor, initiates an **Audit** instance using $(\mathsf{crs}, \phi, id_j)$ as input. We say that the challenger accepts if it accepts in all **Audit** instances.

Going forward, we will treat the above security experiment as occurring in the following two phases, and likewise we will consider the adversary to be of the form $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

- In the first phase, the challenger interacts with $\mathcal{A}_1$ for the **Initialization** and for all **Queries**. We denote $\mathsf{st}^{\mathcal{A}}$ to be $\mathcal{A}$'s internal state at the end of this interaction.
- In the second phase, the experiment enters the **Challenge** phase, and the challenger interacts with $\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})$.

We are now ready to introduce the definition of approximate best-possible recoverability. Throughout, given algorithms $\mathcal{E}$ and $\mathcal{O}$, we use the notation $\mathcal{E}^{\mathcal{O}}$ to mean that the algorithm $\mathcal{E}$ has oracle access to $\mathcal{O}$.

▶ **Definition 3** (Approximate best-possible recoverability). *Let $\epsilon \in (0,1)$ where ϵ is possibly a function in the other parameters. We say that an* iDDA *scheme satisfies ϵ-best-possible-recoverability iff there exist a compression algorithm $\mathcal{C}$, an extractor algorithm $\mathcal{E}$, and a quasi-polynomial function $q(\cdot)$, such that*

- *$\mathcal{C}$'s output $\mathsf{DB}^{\text{short}}$ is at most $\max(B \cdot (n - (1-\epsilon) \cdot S \cdot \mu), 0)$ bits long, and $\mathcal{E}$'s running time is upper bounded by $q(\lambda, t_{\mathcal{A}})$ where $t_{\mathcal{A}}$ denotes $\mathcal{A}$'s maximum running time.*
- *for any non-uniform deterministic polynomial-time algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, any S and μ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every λ,*

$$\Pr\left[\left(\mathsf{DB}^{\text{ext}} \neq \mathsf{DB}^*\right) \wedge (b=1) \;\middle|\; \begin{array}{l} b, \phi, \mathsf{DB}^*, \mathsf{st}^{\mathcal{A}}, tr \leftarrow \mathsf{RecvExpt}^{\mathcal{A}}(1^\lambda, S, \mu) \\ \rho \xleftarrow{\$} \{0,1\}^{|\rho|} \\ \mathsf{DB}^{\text{short}} \leftarrow \mathcal{C}^{\mathcal{A}}(1^\lambda, S, \mu, tr; \rho) \\ \mathsf{DB}^{\text{ext}} \leftarrow \mathcal{E}^{\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})}(1^\lambda, n, S, \mu, \phi, \mathsf{DB}^{\text{short}}; \rho) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where ρ denotes the random coins consumed by the extractor $\mathcal{E}$, which is also shared with the compressor $\mathcal{C}$.

In the above, we use the notation $b, \phi, \mathsf{DB}^*, \mathsf{st}^{\mathcal{A}}, tr \leftarrow \mathsf{RecvExpt}^{\mathcal{A}}(1^\lambda, S, \mu)$ to mean the following: execute the experiment $\mathsf{RecvExpt}^{\mathcal{A}}(1^\lambda, S, \mu)$ with $\mathcal{A}$, and

- *let b denote whether the challenger accepts at the end;*
- *let ϕ denote the digest and let DB^* be the correct database as we enter the* **Challenge** *phase – specifically, ϕ and DB^* can be computed from the initial database DB and the sequence of updates submitted by $\mathcal{A}$ during the* **Initialization** *and* **Query** *phases;*
- *let $\mathsf{st}^{\mathcal{A}}$ be $\mathcal{A}$'s internal state as we enter the* **Challenge** *phase; and*
- *let tr be all of the random coins consumed by the challenger.*

Intuitively, the definition means that if the adversary $\mathcal{A}$ is able to successfully pass the audit and get remuneration on behalf of $\mu \leq \frac{n}{(1-\epsilon)S}$ storage nodes, it must have knowledge of at least $(1-\epsilon) \cdot \mu \cdot S$ blocks of useful information. This information, when combined with an additional $n - (1-\epsilon) \cdot \mu \cdot S$ blocks of information output by the compressor $\mathcal{C}^{\mathcal{A}}$, is sufficient for reconstructing the entire database.

Important special case: recover entire database. When $\mu \cdot S \geq n/(1-\epsilon)$, the definition implies that we must be able to extract the entire database from the μ identities that can pass the audit. In this special case, the compressor $\mathcal{C}^{\mathcal{A}}$'s output is forced to be empty by the definition.

2.2.3 Discussions: Can We Achieve Polynomial-Time Extractability?

Definition 3 allows the extractor to be quasi-polynomial time. One meaningful question is whether we can get polynomial-time extraction – if so, we can simply run the extractor to recover the dataset in practice. We stress that although many works in the proof-of-retrievability [30] and data availability sampling [27] literature claimed to achieve polynomial-time extraction, their extractability notions are much weaker than ours (Definition 3). *If we also adopt the same relaxations to match the nature of the definitions in prior work, we can easily achieve polynomial-time extraction too without rewinding the adversary* – details are provided in the appendix of the full version. For this reason, we do not explicitly define reconstruction in the honest algorithms, since one can just run this extractor to reconstruct.

However, since our paper is aiming to lay the definitional groundwork of decentralized data archival, we choose to take a step back, elucidate the definitional subtleties, and rethink what is the right notion. We believe that our Definition 3 would actually be the desired notion – while our current proof needs a quasi-polynomial time extractor, we leave it as an

interesting open question whether it is possible to achieve polynomial-time extraction under our notion. In particular, we believe that adapting the techniques of Attema et al. [5] to our setting is a promising direction for improving the extractor to expected polynomial time.[3]

We now explain why the relaxations adopted in the prior literature on proofs of retrievability and data availability sampling are not ideal, even though they make polynomial-time extraction possible. Take the work of Hall-Andersen et al. [27] as an example. The differences in the notions are explained below:

- The *soundness* notion of Hall-Andersen et al. [27] (see page 8 of their paper) says that if we invoke the adversary in the audit protocol polynomially many times, then the following holds with probability close to 1: if the adversary succeeds in all of the sessions, then we can extract the dataset from the transcripts collected from all these invocations (and there is no need to rewind the adversary). Their definition is weak in the following sense. Consider an occasionally successful adversary that succeeds in passing the audit with constant probability. Then, the probability that the adversary succeeds in all polynomially many invocations is negligibly small. In other words, *their soundness definition does not provide extractability from such an occasionally successful adversary.* To mitigate this drawback, they provide a slightly strengthened definition called *subset soundness*. Even subset soundness makes a strong assumption on the adversary – it essentially assumes that with probability 1, the adversary will succeed in at least ℓ out of L invocations of audit, where ℓ and L are *a-priori known* polynomials.
- Our notion (Definition 3) follows the definitional paradigm of the knowledge soundness property in the standard zero-knowledge literature. We do not make any a-priori assumption on the success probability of the adversary. To explain our definition, below we focus on the special case where we want to extract the entire database, and we assume that $\mu \cdot S \geq (1+\epsilon)n$, i.e., the number of adversarial nodes μ is large enough such that the combined space of μ nodes can store the entire dataset. Our definition says that the following holds with probability close to 1: as long as all μ adversarial nodes succeed in passing the audit *once*, we can extract the entire dataset by rewinding the adversary. We stress that in our definition, the rewinding is necessary because the transcript length of a single invocation (with all μ nodes) is not long enough to encode the entire dataset, and we cannot simply extract from the transcript with just one invocation.

Another way to understand the comparison is that Hall-Andersen et al. [27]'s definition essentially bakes the extractor into the definition (by invoking the adversary in an audit polynomially many times), and they achieve polynomial-time extraction simply by making a strong assumption on the adversary, i.e., assuming that the adversary will succeed enough times after an *a-priori known* polynomial number of invocations. In the appendix of the full version, we explain how our proofs can be easily extended to show polynomial-time extraction under the same relaxations as in Hall-Andersen et al. [27].

Besides Hall-Andersen et al. [27], other works in this body of literature [30] make some different and non-standard assumptions on the adversary to achieve polynomial-time extraction.

[3] It is also meaningful to compare with the knowledge soundness extractors in the proof systems literature. The extraction in proofs of retrievability [30], data availability sampling [27], and in our work can be viewed as a special case of the knowledge extractor of Kilian [15]. In particular, since our proofs do not involve proving computation, the Probabilistic Checkable Proofs (PCP) in Kilian is replaced with a simpler erasure code. Note strictly polynomial-time extraction for Kilian is not known, and there are reasons to believe that it is impossible [16].

2.3 Security Definition: Replication Security

Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ denote the adversary's algorithm, where $\mathcal{A}_1$ participates in the **Initialization** and **Query** phases of the PoRepExpt to be defined, and $\mathcal{A}_2$ participates in the **Challenge** phase. Now, we define the following security experiment.

Experiment $\mathsf{PoRepExpt}^{\mathcal{A}}(1^\lambda, S, \mu)$:

- **Initialization**. Same as in the RecvExpt experiment earlier, where $\mathcal{A}_1(1^\lambda, S, \mu)$ interacts with the challenger.
- **Queries.** Same as in the RecvExpt experiment earlier, where $\mathcal{A}_1$ continues to interact with the challenger. At the end of the query phase, $\mathcal{A}_1$ outputs some state $\mathsf{st}^{\mathcal{A}}$ to be passed to $\mathcal{A}_2$.
- **Challenge.** $\mathcal{A}_2$ receives $\mathsf{st}^{\mathcal{A}}$ as input, and outputs μ challenge identities $id_1, \ldots, id_\mu$. The challenger picks a random $id \in \{id_1, \ldots, id_\mu\}$, and invokes an **Audit** instance with $\mathcal{A}_2$, which acts on behalf of identity id. The adversary is said to win this game if it passes the audit.

▶ **Definition 4** (Replication security)**.** *Let $\epsilon \in (0, 1)$. We say that a decentralized data archival scheme satisfies ϵ-replication-security iff for any non-uniform deterministic polynomial-time adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that $\mathcal{A}_1$ is restricted to outputting a state $\mathsf{st}^{\mathcal{A}}$ of space at most $\mu \cdot \alpha \cdot BS$, then the probability that $\mathcal{A}$ wins the above $\mathsf{PoRepExpt}^{\mathcal{A}}(1^\lambda, S, \mu)$ game is at most $\alpha - \epsilon$.*

Intuition. Intuitively, replication security says that if an adversary wants to get α times the fair reward in expectation, it must be consuming roughly α times the space, and this must hold even when the original data itself is compressible. We now elaborate on how to understand the definition. Suppose the adversary dedicates $\alpha\mu S$ blocks of space where S is the space of an honest node. We expect that in every period, the adversary should get roughly $\alpha\mu$ times the fair reward. However, the adversary can allocate $\alpha\mu S$ blocks of space among its μ nodes in various ways: it can allocate S space for $\alpha \cdot \mu$ nodes and 0 space for the remaining nodes, but it can also equally allocate $\alpha \cdot S$ space to each of the μ nodes. Regardless, the adversary should not get noticeably more than $\alpha \cdot \mu$ times the fair reward. Our replication security definition captures this intuition by randomly sampling a challenge node among the μ specified nodes, which is effectively an "averaging argument" to capture the idea that "no matter how the adversary allocates its space, it should get no more than $\alpha \cdot \mu$ times the fair reward in expectation". Another reason why we randomly sample a challenge node in the definition rather than challenging all of them is because in practice, it may make sense to randomly sample the time at which each node is challenged. If the challenge times are known ahead of time, then the adversary can reconstruct the storage right before the challenge and delete the space afterwards.

Our replication security notion implies an ϵ-Nash-equilibrium, that is, a node with a fixed amount of space cannot get ϵ fraction more rewards than behaving honestly and contributing all of its space towards archiving the dataset. Because honest behavior is an equilibrium, it implies that that in the equilibrium state, any node claiming rewards commensurate with contributing αn blocks of space is actually storing α copies of the database.

3 Informal Technical Roadmap

In this section, we give an informal description of the ideas behind our construction. A formal description is provided in the full version of the paper.

To understand the technicalities, let us begin with a couple strawman solutions, and gradually work our way towards the final solution.

3.1 Inefficient Strawman

We begin with a strawman scheme that indeed satisfies the desired security notions, despite being inefficient.

Underlying static construction. First, if the database were static, we can employ the following idea.

- *Preprocess and publish.* The trusted data publisher computes an erasure code over the original database DB, resulting in $\overline{\mathsf{DB}}$. It then computes and publishes the Merkle digest denoted ϕ^{DB} of $\overline{\mathsf{DB}}$.
- *Store.* Every storage node with roughly S blocks of space uses a random oracle $G(id, \cdot)$ to sample S indices. It then downloads $\overline{\mathsf{DB}}[S]$ along with the relevant Merkle opening proofs, and computes a replication encoding of $\overline{\mathsf{DB}}[S]$ henceforth called the node's *shard*. Specifically, we will use the PoRep scheme of Pietrzak [34]. The resulting replication encoding has S blocks, and it provides the guarantee that the encoding is incompressible if the node later wants to pass the audit within bounded time.
  The node additionally computes a corresponding Merkle digest ϕ^{shard} of its replication-encoded shard, and a succinct correct proof π attesting to the fact that ϕ^{shard} is computed correctly w.r.t. $G(id, \cdot)$ and ϕ^{DB}. Finally, the node stores the following information:
  1. its shard, the shard's Merkle digest ϕ^{shard}, as well as a proof of correctness π of the digest ϕ^{shard}, and
  2. the Merkle openings of all (replication-encoded) blocks in the shard.
- *Audit.* An auditor samples $\kappa = \omega(\log \lambda)$ random challenge indices among $[S]$ and asks the storage node to open the replication-encoded blocks at these positions. The node responds with the challenged positions, along with ϕ^{shard}, π, and the Merkle opening proofs of the opened positions w.r.t. ϕ^{shard}. The auditor accepts if π verifies and all Merkle opening proofs verify.

Throughout the paper, we assume that the block size B is at least polylogarithmically large, such that the space required for storing metadata (e.g., digests and opening proofs) is absorbed by the block storage.

▶ Remark 5 (Regarding the succinct proof of correctness). The aforementioned succinct proof of correctness can be computed using a Succinct Non-interactive ARgument of Knowledge (SNARK) scheme. The SNARK is undesirable not only due to the extra computational costs, but also because we need a SNARK proof over computations that involve random oracle queries. Recent work [6] has shown that constructing such a relativized SNARK is impossible. We discuss how to get rid of the SNARK proof in Section 3.4.

Making it dynamic: inefficient approach. Now, if the database is evolving over time, the most naïve approach is to rerun the static scheme from scratch with every update. The resulting scheme indeed satisfies ϵ-best-possible recoverability as well as ϵ-replication-security for an arbitrarily small $\epsilon \in (0, 1)$. Unfortunately, for each update, the scheme would incur $\widetilde{O}(B \cdot N)$ cost for the publisher and $O(B \cdot S)$ cost for each storage node, where N is the maximum data size.

We ask the natural question: can we reduce the cost per update to $\widetilde{O}(1) \cdot B$?

3.2 How to Apply a Hierarchical Data Structure: the Space Allocation Problem

To answer the above question, the first idea that comes to mind is to apply the hierarchical data structure of Bentley and Saxe [9], which turns a static data structure into a dynamic one. This approach also draws inspiration from prior works on dynamic proofs of retrievability [13,38].

Unfortunately, the hierarchical data structure does not generically work for any cryptographic scheme. In particular, as explained below, both our security definitions and our underlying cryptographic building blocks lack the appropriate compositional guarantees needed for a *blackbox* application of the hierarchical data structure. To see this, it helps to go over a couple strawman ideas.

Review: publisher's hierarchical data structure. With Bentley and Saxe's techniques [9], the publisher can maintain a hierarchy of levels numbered $0, 1, \ldots, L = O(\log N)$, respectively, where each level $\ell \in \{0, 1, \ldots, L\}$ is either an erasure code of 2^ℓ blocks, or empty.

Every time a new data block arrives, we find the smallest empty level denoted ℓ^*, and merge all smaller levels as well as the newly arriving block into level ℓ^*, by recomputing an erasure code of these blocks. The levels smaller than ℓ^* now become empty. Further, suppose that the data publisher publishes a Merkle digest of each level. Assuming that the block size B is larger than the size of a Merkle opening proof, and the erasure code has constant rate, then the amortized publisher cost for maintaining this hierarchical data structure can be as small as $O(B \log N)$ if we use a special updatable erasure code proposed in prior work [38].

The space allocation question. The idea is for each storage node with approximately S blocks of space to choose s_ℓ random erasure-coded blocks per non-empty level for its shard, such that $\sum_\ell s_\ell = S$. As before, this selection can be made with the help of a random oracle $G(id, \cdot)$. During the audit, the auditor will challenge $\mu = \omega(\log \lambda)$ random indices per level.

The challenging question is: how do we allocate the local space S among the levels? To understand the subtleties here, it helps to look at a couple naïve approaches:

- (Idea 1) *Uniform sampling rate: prone to selective censorship.* The first idea is to use a uniform sampling rate. Specifically, let $p = S/n$ where S is the local space provisioning and n is the current database size – for a growing database, we can recompute the sampling rate p whenever the database size has reached $(1 + o(1)) \cdot n_{\text{prev}}$ where n_{prev} denotes the database size when p was last calculated. Now, a node would sample every block in any level with uniform probability p, so that in expectation, it will get S blocks for its shard.

  Unfortunately, this natural approach is fundamentally flawed. Under this approach, a node would end up sampling $\Theta(S)$ blocks in expectation for the largest level. However, for any constant-sized level (say, level 0), the fraction of nodes required to store some block of that level is only $\Theta(S/n)$. This means that if the adversary selectively deletes the small fraction of identities assigned to store some block of level 0, it can completely wipe out the data belonging to level 0! Another way to think of the same attack is the following. As mentioned, it could be that the adversary controls all the pseudonyms that are contributing and requesting rewards. In this case, if the adversary chooses only identities that are not assigned any block of level 0 (e.g., through rejection sampling), it can successfully wipe out level 0 altogether while still being able to pass the audits.
- (Idea 2) *Same number of blocks per level: causes space waste.* To fix the problem with idea 1, we can increase the sampling rate of the smaller levels. A natural idea is to sample the same number of blocks per level regardless of the level's size. In other words,

a node samples $s_\ell = S/\widehat{L}(n)$ blocks for every non-empty level, where $\widehat{L}(n)$ denotes the number of non-empty levels under a size-n database. Unfortunately, this approach suffers from a different problem partly because the smaller levels effectively replicate their data much more abundantly than the larger levels, leading to a waste problem. In particular, ϵ-best-possible recoverability requires that when μ nodes pass the audit and the total space provisioned satisfies $\mu \cdot S \approx (1+\epsilon)n$, we should be able to extract the entire dataset. However, when μ is chosen to ensure recoverability of the largest level L, i.e., $\mu \cdot s_\ell \approx (1+\epsilon) \cdot 2^L$, the total space provisioned $\mu \cdot S$ could exceed the data size by a logarithmic factor, that is, $S \cdot \mu = \Omega(n \log n)$. In other words, this approach would require a total space provisioning of $\Theta(n \log n)$ rather than $(1+\epsilon)n$ to recover a database of size n.

More fundamentally, Idea 2 fails partly because our *security definition itself lacks compositional guarantees*. Specifically, Idea 2 can in fact be shown to satisfy ϵ-best-possible recoverability *for each individual level*, as long as we select the parameter S and the rate of the erasure code satisfy some mild assumptions. Unfortunately, it is not the case that if each individual level satisfies ϵ-best-possible recoverability, the union of all levels would also satisfy the same notion. Partly, this is because the number of nodes μ needed for the space provisioning per level to roughly match the level's size can vary across the levels!

A hybrid space allocation scheme. We resolve the aforementioned challenges by using a hybrid of the aforementioned ideas. Specifically, we divide the levels into two categories: the biggest $2 \log \log n$ levels numbered $L - 2\log\log n + 1$ to L are called the *large levels*, and all remaining levels are called the *small levels*. In our formal description later, we generalize the parameter $2 \log \log n$ to more general forms, but for clarity, we simply use $2 \log \log n$ in the informal roadmap. By renaming, we may assume that each node has $(1 + o(1))S$ blocks of space rather than S for some suitable sub-constant function $o(1)$. We will allocate the $(1 + o(1))S$ blocks of space among the levels as follows:

- *Large levels.* We dedicate S blocks of space in aggregate to the large levels. Among the large levels, we adopt a uniform sampling rate. In other words, a large level $i+1$ occupies twice as much space as level i.
- *Small levels.* We dedicate $S \cdot o(1)$ blocks of space in aggregate to the small levels. Further, this space is divided evenly across the small levels.

This hybrid approach achieves the best of both worlds. First, by having a higher sampling rate in the smaller levels, we prevent the aforementioned selective censorship attack. Second, since nearly all the space is allocated to the large levels, the space waste caused by the smaller levels is restricted to $S \cdot o(1)$. Moreover, the $o(1)$ factor loss can be absorbed into the ϵ-slack already permitted in our security definition.

One remaining technicality is how to formally argue resilience against the selective censorship attack mentioned earlier. In particular, if for some level s_ℓ, each storage node samples only $s_\ell = 1$ block to store, then an adversary can carefully select a set of identities that cause 2/3 of the (erasure-coded) blocks to be lost. Specifically, the adversary can use rejection sampling to select only identities that are assigned a block from the first 1/3. Intuitively, when s_ℓ is larger, erasing 2/3 of the blocks appears much harder, since only a negligible fraction of ids avoid hitting any specific 1/3 of the blocks.

To formalize this intuition, we consider an adversary that can make at most polynomially many queries to $G(id, \cdot)$. After these queries, it selects a subset of μ queried identities denoted $id_1, \ldots, id_\mu$ to minimize the union $G(id_1, \cdot) \cup \ldots \cup G(id_\mu, \cdot)$. We prove that with all

but negligible probability, $|G(id_1, \cdot) \cup \ldots \cup G(id_\mu, \cdot)| \geq \min(2^\ell, (1-\epsilon) \cdot s_\ell \cdot \mu)$, as long as *i)* the number of blocks sampled s_ℓ is at least polylogarithmically large, and *ii)* the redundancy (i.e., inverse rate) of the erasure code is a suitably large constant. In particular, this implies that as long as $(1-\epsilon) \cdot s_\ell \cdot \mu \geq 2^\ell$, then the union of any μ nodes' shards are sufficient for reconstructing level ℓ which contains 2^ℓ blocks – even when the identities are adversarially chosen.

Of course, our actual proof is more complicated than the above. In the actual proof, the above combinatorial reasoning is embedded in some extraction argument that takes into account the fact that even when a node passes the audit, it may not be storing all blocks belonging to its shard. We defer the details to Section 9 of the full version.

The careful reader may also observe that a straightforward instantiation of our hybrid space allocation idea would incur roughly $\widetilde{O}(S) \cdot B$ amortized cost per update for a storage node. However, later in Section 3.4, we will discuss some additional tricks that bring this cost down to $\widetilde{O}(1) \cdot B$.

3.3 Handling Compositional Challenges

Lack of composition in the underlying replication code. Although the new hybrid space allocation appears to address the aforementioned issues, we are not aware of any method to formally prove its security. Specifically, one important challenge we face is that the underlying replication encoding scheme [34] lacks compositional guarantees.

Pietrzak [34] only proved the incompressibility (subject to answering challenges quickly) of his replication encoding scheme in a standalone setting. More specifically, imagine that there is a single database, and we use the digest of the database to seed the hash function used to compute the replication code. Then, the resulting replication code is incompressible subject to answering challenges quickly.

In our application, there is a separate instance of the replication encoding per level. The most straightforward approach is for each level to use the level's own digest (along with the storage node's *id*) to seed its own hash function. With this approach, however, the adversary in our security experiment would be able to choose the data contents of the smaller levels to depend on the replication codes of the larger levels. This is because in our security experiment, the adversary chooses the updates adaptively over time. So when it chooses the blocks that go into the smaller levels, the replication codes of the larger levels are already available.

Unfortunately, Pietrzak's proofs [34] fall apart in such a setting when multiple instances are composed and some instances' data can depend on other instances' replication code. Upon a closer examination, Pietrzak's proof has the following blueprint – henceforth let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_1$ is the adversary that outputs some database DB and an adversarial replication encoding denoted $\mathsf{st}^{\mathcal{A}}$ of DB, and $\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})$ is the adversary interacting with the auditor.

- First, he shows that if $\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})$ can answer challenges quickly, then there is a winning strategy to an underlying pebbling game. Specifically, by placing some initial labels on the depth-robust graph, the adversary can pebble almost all vertices in a small number of steps.
- Second, he analyzes the underlying pebbling game and argues that for any winning strategy, the number of initial pebbles must be large.
- Finally, he shows that if $\mathsf{st}^{\mathcal{A}}$ is short, and the number of initial pebbles is large, then one can construct an encoding scheme to compress the random oracle $H(\phi^{\mathrm{DB}}, \cdot)$ where ϕ^{DB} is the digest of the challenge database, thus contradicting Shannon's theorem that

random strings are incompressible. Intuitively, every initial pebble corresponds to a location $\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})$ can predict in $H(\phi^{\mathrm{DB}}, \cdot)$. As a result, we need not record these predicted positions in the encoded string, thus achieving compression. The encoded string consists of the short $\mathsf{st}^{\mathcal{A}}$, $H(\phi^{\mathrm{DB}}, \cdot)$ at all non-predicted locations, and a small amount of metadata needed for extracting from $\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})$ the predicted locations.

The subtlety in the proof lies with the decoder. For technical reasons, the decoder needs to know the database to successfully extract $H(\phi^{\mathrm{DB}}, \cdot)$ at the predicted locations. In a standalone setting, before running $\mathcal{A}_2$ to perform decoding, the decoder first runs $\mathcal{A}_1$ till the point it first submits ϕ^{DB} to extract the database – henceforth this is called the preparation stage. If the digest ϕ^{DB} is also computed from a random oracle, then except with negligible probability, $\mathcal{A}_1$ cannot have queried $H(\phi^{\mathrm{DB}}, \cdot)$ yet before revealing the database. In our composed setting, the same proof strategy fails, since $\mathcal{A}_1$ will submit the different level's data incrementally. This means that before submitting the data in level 1, $\mathcal{A}_1$ may already start querying $H(\phi_0^{\mathrm{DB}}, \cdot)$ yet where ϕ_0^{DB} is the digest of the 0-th level. Unfortunately, the decoder has no way of answering queries to $H(\phi_0^{\mathrm{DB}}, \cdot)$ yet in the preparation stage, without having run $\mathcal{A}_2$ to perform the decoding.

Although the literature also comes with some other replication encoding candidates [22,23], to the best of our knowledge, no known scheme provides the compositional guarantees we desire.

Our idea. One way to solve the problem is to devise a replication coding scheme with the desired compositional guarantees, where the data of some instances may depend on the replication code of other instances. Unfortunately, we are not aware of any existing tools that can be used to achieve this: existing techniques for proving space-time tradeoffs through direct incompressibility arguments are highly involved and apply to extremely limited settings [19,20]. While some other proof techniques [1,2,18,25,39] have been shown to prove space-time tradeoffs, they do not produce meaningful results in our setting.

Fortunately, we devise a method that side-steps this problem. Whenever a new erasure-coded level is rebuilt in the hierarchical data structure, we ask a storage node to recompute its replication codes for all levels, using the union of all levels' digests $(\phi_0, \ldots, \phi_L)$ along with the node's *id* as the seed to the random oracle.

At first sight, this approach comes with additional computational overhead on the storage node. Specifically, the computational cost per update is at least $S \cdot B$. However, in Section 3.4, we discuss additional tricks to asymptotically reduce the computational overhead, and achieve $\widetilde{O}(1) \cdot B$ amortized cost per update.

3.4 Further Improvements

Achieving $\widetilde{O}(1) \cdot B$ amortized cost for a storage node. So far, we have a candidate scheme but each storage node must pay at least $S \cdot B$ download bandwidth and computational cost per update. We propose a couple additional tricks to bring this cost down to $\widetilde{O}(1) \cdot B$. For example, when S is roughly $\sqrt{n}$ (see also Section 7.1 of the full version), these tricks bring significant cost savings. We stress that *it is inherently unavoidable that each node must pay at least constant cost per update subject to our definitions – in this sense, our update costs are nearly optimal.* Specifically, if 1% of the nodes do nothing upon an update, it means that they have no information about the new block. Now, if the adversary selectively corrupts the 99% remaining nodes, it can selectively erase this new block from the universe even if the 1% nodes' combined storage is already large enough to store the entire dataset.

1. *Small* $\Longrightarrow$ {*tiny, mini, small*}: We further divide the small levels into tiny, mini, and small levels. For the tiny levels each containing at most $\kappa = \omega(\log \lambda)$ blocks, the node simply stores the original data blocks (without any encoding), and *all* of the blocks are challenged during the audit. For the mini levels each containing between κ and $\Theta(S/\log^2 n)$ blocks, the node stores *all* encoded blocks belonging to the level (without using the random oracle G to subsample), and $\kappa = \omega(\log \lambda)$ of them will be challenged during the audit. The small levels are treated in the same way as before.
2. *Seed with an aggregate digest only for the large levels:* Instead of making all levels' replication codes use the aggregate digest $(\phi_0, \ldots, \phi_L)$ as the seed, we have only the large levels use the aggregate digest $\{\phi_\ell\}_{\ell \in \text{large}}$, and the small levels use the level's own digest as the seed. Recall that the large levels occupy $1 - o(1)$ fraction of the local space, this modification does not affect our ϵ-replication security since the $o(1)$ loss can be absorbed into the arbitrarily small constant slack $\epsilon \in (0, 1)$.

A more detailed description and analysis of these optimizations are provided in Section 6.2 of the full version.

Removing the SNARK proof. Recall that so far, in our underlying static construction, we rely on a SNARK proof to attest to the correctness of the shard's digest ϕ^{shard} w.r.t. the database's digest ϕ^{DB} and the indices selected by the shard $G(id, \cdot)$. Using a SNARK, however, comes with a couple drawbacks. First, it incurs additional costs of cryptographic computation. Second, since the replication encoding is computed using a random oracle (RO), we would end up computing a SNARK over computations that involve RO queries – this is undesirable due to impossibilities of relativized succinct arguments in the RO model.

In our final construction, we avoid this SNARK proof by checking correctness at a few challenged locations, resulting in a proof of *approximate* correctness (rather than *strict* correctness). Further, we make our approach non-interactive by having the node sample the challenges itself through a random oracle (commonly known as the Fiat-Shamir paradigm). The fact that we only have approximate correctness introduces additional technicalities in our proof. Note that in comparison, Pietrzak's proof works only if the encoder is honest. Nonetheless, we show that approximate correctness is sufficient for proving our security notions. We defer the details to the subsequent formal technical sections.

Extensions. In Section 7 of the full version, we discuss a couple of extensions. Specifically, we show how to support non-uniform space provisioning among nodes, ensuring that a node with k times the space need not incur k times the update and audit costs. We also show when provided with a trusted hash digest of the original dataset ϕ_{orig} (e.g., from the blockchain's consensus layer), how to instantiate the publisher without any trust, by relying on an Incremental Verifiable Computation (IVC) scheme to incrementally compute succinct proofs that vouch for the correctness for the digests of the erasure-coded hierarchical data structure.

References

1 Akshima, David Cash, Andrew Drucker, and Hoeteck Wee. Time-space tradeoffs and short collisions in merkle-damgard hash functions. In *CRYPTO*, 2020.

2 Akshima, Siyao Guo, and Qipeng Liu. Time-space lower bounds for finding collisions in merkle-damgård hash functions. In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part III*, volume 13509 of *Lecture Notes in Computer Science*, pages 192–221. Springer, 2022. doi:10.1007/978-3-031-15982-4_7.

3 Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin, and Ismail Khoffi. Fraud and data availability proofs: Detecting invalid blocks in light clients. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II*, pages 279–298, Berlin, Heidelberg, 2021. Springer-Verlag. doi:10.1007/978-3-662-64331-0_15.

4 Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In *ASIACRYPT*, 2009.

5 Thomas Attema, Michael Klooß, Russell W. F. Lai, and Pavlo Yatsyna. Adaptive special soundness: Improved knowledge extraction by adaptive useful challenge sampling. *IACR Cryptol. ePrint Arch.*, page 2038, 2024. URL: https://eprint.iacr.org/2024/2038.

6 Annalisa Barbara, Alessandro Chiesa, and Ziyi Guan. Relativized succinct arguments in the ROM do not exist. *IACR Cryptol. ePrint Arch.*, page 728, 2024. URL: https://eprint.iacr.org/2024/728.

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

8 Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.14.

9 Jon Louis Bentley and James B Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980. doi:10.1016/0196-6774(80)90015-2.

10 Dan Boneh, Joachim Neu, Valeria Nikolaenko, and Aditi Partap. Data availability sampling with repair. Cryptology ePrint Archive, Paper 2025/1414, 2025. URL: https://eprint.iacr.org/2025/1414.

11 Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: theory and implementation. In *CCSW*, 2009.

12 Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *Proceedings of the 19th International Conference on Distributed Computing*, DISC'05, pages 503–504, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11561927_42.

13 Nishanth Chandran, Bhavana Kanukurthi, and Rafail Ostrovsky. Locally updatable and locally decodable codes. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, volume 8349 of *Lecture Notes in Computer Science*, pages 489–514. Springer, 2014. doi:10.1007/978-3-642-54242-8_21.

14 Arunima Chaudhuri, Sudipta Basak, Csaba Kiraly, Dmitriy Ryajov, and Leonardo Bautista-Gomez. On the design of ethereum data availability sampling: A comprehensive simulation study, 2024. doi:10.48550/arXiv.2407.18085.

15 Alessandro Chiesa, Marcel Dall'Agnol, Ziyi Guan, Nicholas Spooner, and Eylon Yogev. Untangling the security of kilian's protocol: Upper and lower bounds. In *Theory of Cryptography: 22nd International Conference, TCC 2024, Milan, Italy, December 2–6, 2024, Proceedings, Part I*, pages 158–188, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-3-031-78011-0_6.

16 Alessandro Chiesa, Ziyi Guan, and Yuetian Wu. Unpublished manuscript. Private conversation with Ziyi Guan, 2025.

17 Seunghyun Cho, Eunyoung Seo, and Young-Sik Kim. Locally recoverable data availability sampling. Cryptology ePrint Archive, Paper 2025/1851, 2025. URL: https://eprint.iacr.org/2025/1851.

18 Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John P. Steinberger. Random oracles and non-uniformity. In *EUROCRYPT (1)*, pages 227–258. Springer, 2018. doi:10.1007/978-3-319-78381-9_9.

19 Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and prgs. In *Advances in Cryptology – CRYPTO 2010*, pages 649–665, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-14623-7_35.

20 Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In *EUROCRYPT (2)*, pages 473–495. Springer, 2017. doi:10.1007/978-3-319-56614-6_16.

21 Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *Theoretical Cryptography Conference (TCC)*, 2009.

22 Ben Fisch. Poreps: Proofs of space on useful data. *IACR Cryptol. ePrint Arch.*, page 678, 2018. URL: https://eprint.iacr.org/2018/678.

23 Ben Fisch. Tight proofs of space and replication. In *Advances in Cryptology – EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part II*, pages 324–348, Berlin, Heidelberg, 2019. Springer-Verlag. doi:10.1007/978-3-030-17656-3_12.

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

25 Nick Gravin, Siyao Guo, Tsz Chiu Kwok, and Pinyan Lu. Concentration bounds for almost k-wise independence with applications to non-uniform security. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '21, pages 2404–2423, USA, 2021. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611976465.143.

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

27 Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Foundations of data availability sampling. *IACR Commun. Cryptol.*, 1(4):34, 2024. doi:10.62056/A09QUDHDJ.

28 Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. FRIDA: data availability sampling from FRI. In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VI*, volume 14925 of *Lecture Notes in Computer Science*, pages 289–324. Springer, 2024. doi:10.1007/978-3-031-68391-6_9.

29 James Hendricks, Gregory R. Ganger, and Michael K. Reiter. Verifying distributed erasure-coded data. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, pages 139–146, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1281100.1281122.

30 Ari Juels and Burton Kaliski. Pors: proofs of retrievability for large files. In *ACM CCS*, 2007.

31 Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010. doi:10.1007/978-3-642-17373-8_11.

32 A. Kupcu. Efficient cryptography for the next generation secure cloud. PhD thesis, Brown University, 2010.

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

34 Krzysztof Pietrzak. Proofs of catalytic space. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 59:1–59:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.59.

35 Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *Asiacrypt*, 2008.

36 Hovav Shacham and Brent Waters. Compact proofs of retrievability. *J. Cryptol.*, 26(3):442–483, July 2013. doi:10.1007/S00145-012-9129-2.

37 Elaine Shi, Rose Silver, and Changrui Mu. Decentralized data archival: New definitions and constructions. Cryptology ePrint Archive, Paper 2025/969, 2025. URL: https://eprint.iacr.org/2025/969.

38 Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

39 Dominique Unruh. Random oracles and auxiliary input. In *Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO'07, pages 205–223, Berlin, Heidelberg, 2007. Springer-Verlag. doi:10.1007/978-3-540-74143-5_12.

40 Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943, 2018. doi:10.1109/SP.2018.00060.

41 Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. DispersedLedger: High-Throughput byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 493–512, Renton, WA, April 2022. USENIX Association. URL: https://www.usenix.org/conference/nsdi22/presentation/yang.