Decentralized Data Archival: New Definitions and Constructions

Elaine Shi
CMU
elainershi@gmail.com

Rose Silver
CMU
rosesilv@andrew.cmu.edu

Changrui Mu*
NUS, CMU
changrum@andrew.cmu.edu

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.

*Author ordering is randomized.

Contents

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. Further, 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 an 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 attaining rewards commensurate with αn copies of the database, it has indeed dedicated roughly αn amount of space. Inspired by prior works [Fis19, Fis18, Pie19], 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) [JK07, Kup10, SW08, DVW09, BJO09, AKK09, SW13, SSP13, CKO14], data availability sampling (DAS) [HSW24a, HSW24b, CBK+24, ABSBK21, GXQZ25], proofs of replication (PoRep) [Fis18, Fis18, Pie19][1], and verifiable information dispersal (VID) [FLLY24, NNT23, BBC+24, YPA+22, CT05, HGR07], 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 [JK07, Kup10, SW08, DVW09, BJO09, AKK09, SW13, SSP13, CKO14], DAS [HSW24a, HSW24b, CBK+24], and PoRep [Fis18, Fis18, Pie19].
- A separate instance of the scheme is applied on a per-block basis — an assumption made by prior works on DAS [HSW24a, HSW24b, CBK+24], and VID [FLLY24, NNT23, BBC+24, YPA+22, CT05, HGR07]. 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 [FLLY24, NNT23, BBC+24].
- 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.

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

[1]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.

provisioning required to contribute is only polylogarithmic. Further, 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 1.1 (Main theorem)**.** *Assume the random oracle model. Let $\epsilon \in (0, 1)$ be an arbitrarily small constant. Further, suppose $B \geq \mathsf{poly}\log\lambda$ and $S \geq \mathsf{poly}\log\lambda$ 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 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 (Section 2.1 and Section 5). 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 [BS80] 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 [Fis19, Fis18, Pie19] 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 2.2 for details. For the second challenge, we are not aware of any approach to extend the proofs in earlier works [Fis19, Fis18, Pie19] 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 [DTT10, DGK17]. While some other proof techniques [Unr07, CDGS18, ACDW20, GGKL21, AGL22] 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 2.4, 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 [SSP13, BBC+24]. 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 [BBC+24, SSP13], 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 [Fis19, Fis18, Pie19] never considered this natural compositional notion is exactly because they (implicitly) considered a setting with 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 for details. Our paper focuses on append-only updates, so a natural direction is to extend the scheme to support other types of updates.

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) [HSW24a, HSW24b, CBK+24, ABSBK21, GXQZ25] and and Verifiable Information Dispersal (VID) [FLLY24, NNT23, BBC+24, YPA+22, CT05, HGR07] 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 barier 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) [JK07, Kup10, SW08, DVW09, BJO09, AKK09, SSP13, CKO14, SW13], 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 [SSP13, CKO14].

Data availability sampling [HSW24a, HSW24b, CBK+24, BNNP25, CSK25] 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) [FLLY24, NNT23, BBC+24], 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) [Fis19, Fis18, Pie19] 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 [Fis19, Fis18, Pie19] 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) [KZG10, WTS+18, BBHR18]. 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 [KZG10, WTS+18, BBHR18]. In comparison, our approach has only $B \cdot \widetilde{O}(1)$ amortized publisher cost per update.

2 Informal Technical Roadmap

In this section, we give an informal description of the ideas behind our construction. A formal description is provided in the subsequent technical sections.

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

2.1 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 [Pie19]. 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 1 (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 [BCG24] has shown that constructing such a relativized SNARK is impossible. We discuss how to get rid of the SNARK proof in Section 2.4.

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 $\widetilde{O}(1) \cdot B$?

2.2 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 [BS80], which turns a static data structure into a dynamic one. This approach also draws inspiration from prior works on dynamic proofs of retrievability [SSP13, CKO14].

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 [BS80], 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 [SSP13].

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 *id*s 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.

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 2.4, we will discuss some additional tricks that bring this cost down to $\widetilde{O}(1) \cdot B$.

2.3 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 [Pie19] lacks compositional guarantees.

Pietrzak [Pie19] 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 [Pie19] 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 $H(\phi^{\text{DB}}, \cdot)$ 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 $H(\phi^{\text{DB}}, \cdot)$. 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 [Fis19, Fis18], 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 [DTT10, DGK17]. While some other proof techniques [Unr07, CDGS18, ACDW20, GGKL21, AGL22] 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 2.4, we discuss additional tricks to asymptotically reduce the computational overhead, and achieve $\widetilde{O}(1) \cdot B$ amortized cost per update.

2.4 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), 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.

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, we discuss a couple 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.

3 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.

3.1 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:

- $\mathsf{crs} \leftarrow \mathbf{Setup}(1^\lambda)$: a setup algorithm that takes as input the security parameter 1^λ and outputs a common reference string crs.
- $(\phi, \mathsf{st}^0) \leftarrow \mathbf{Init}(\mathsf{crs}, S, \mathsf{DB})$: 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.
- $(\mathsf{st}^0, \mathsf{st}^{id}) \leftarrow \mathbf{Join}(\mathsf{st}^0, id)$: 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.
- $(\phi, \mathsf{st}^0, \{\mathsf{st}^{id}\}_{id \in \mathsf{IDset}}) \leftarrow \mathbf{Update}\Big((\mathsf{st}^0, \mathsf{upd}), \{\mathsf{st}^{id}\}_{id \in \mathsf{IDset}}\Big)$: 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.
- $b \leftarrow \mathbf{Audit}((\mathsf{crs}, \phi, id), \mathsf{st}^{id})$: 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, 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 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 3.3 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 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.

3.2 Security Definition: Approximate Best-Possible Recoverability

3.2.1 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.

3.2.2 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 occuring 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 1 (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.

3.2.3 Discussions: Can We Achieve Polynomial-Time Extractability?

Definition 1 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 [JK07] and data availability sampling [HSW24a] literature claimed to achieve polynomial-time extraction, their extractability notions are much weaker than ours (Definition 1). *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* — see Appendix A for details. 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 1 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. [AKLY24] to our setting is a promising direction for improving the extractor to expected polynomial time.[2]

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. [HSW24a] as an example. The differences in the notions are explained below:

- The *soundness* notion of Hall-Andersen et al. [HSW24a] (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 1) 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. [HSW24a]'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 Appendix A, we explain how our proofs can be easily extended to show polynomial-time extraction under the same relaxations as in Hall-Andersen et al. [HSW24a].

Besides Hall-Andersen et al. [HSW24a], other works in this body of literature [JK07] make some different and non-standard assumptions on the adversary to achieve polynomial-time extraction.

[2] It is also meaningful to compare with the knowledge soundness extractors in the proof systems literature. The extraction in proofs of retrievability [JK07], data availability sampling [HSW24a], and in our work can be viewed as a special case of the knowledge extractor of Kilian [CDG+24]. 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 [CGW25].

3.3 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 2 (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.

4 Preliminaries

4.1 Depth-Robust Graphs

Definition 3 ((e, d)-depth-robust graph [EGS75, ABP18])**.** Let $e, d \in [0, 1]$. A directed acyclic graph (DAG) $\mathsf{DRG} = (V, E)$ on $|V| = n$ nodes is said to be (e, d)-depth-robust if after removing any subset of $e \cdot n$ nodes, there remains a path of length $d \cdot n$.

In this work, we will use the depth-robust graphs (DRG) proposed by Alwen et al. [ABP18], where each vertex has in-degree $O(\log n)$. Pietrzak [Pie19] showed how to use such a DRG to construct proof of space. Specifically, suppose a node wants to prove that it is dedicating n amount of space. Proof of space guarantees that if an adversary is restricted to αn space for some $\alpha \in (0,1)$, then it can only answer roughly $\alpha - \zeta$ fraction of challenges within n rounds for some arbitrarily small constant $\zeta \in (0,1)$.

4.2 Erasure Code

Henceforth, we use Σ to denote the alphabet associated with the original message, and we use $\overline{\Sigma}$ to denote the alphabet associated with the codeword. Let $\ell(n)$ denote the length of the encoding of a length-n message. An erasure code over some finite alphabet Σ has the following algorithms:

- $\mathsf{C} \leftarrow \mathbf{Encode}(\mathsf{msg})$: a deterministic algorithm that on input of some message $\mathsf{msg} \in \Sigma^n$, outputs a codeword $\mathsf{C} \in \overline{\Sigma}^{\ell(n)}$.
- $\mathsf{msg} \leftarrow \mathbf{Decode}(\mathsf{C})$: a deterministic algorithm which on input of some encoded $\mathsf{C} \in \{\overline{\Sigma} \cup \{\perp\}\}^{\ell(n)}$ performs decoding and outputs msg. Note that some entries in the input C may be dropped and replaced with $\perp$.

Correctness. The erasure code satisfies correctness iff the following holds: for any $\mathsf{msg} \in \Sigma^n$, let $\mathsf{C} \leftarrow \mathbf{Encode}(\mathsf{msg})$, and let $\mathsf{C}' \in \{\overline{\Sigma} \cup \perp\}^{\ell(n)}$ be any vector that agrees with C in at least n positions whereas all remaining positions are $\perp$, then $\mathbf{Decode}(\overline{\mathbf{C}}') = \mathsf{msg}$.

Rate and redundancy of the code. The *rate* of the code is defined to be $n/\ell(n)$. The *redundancy* is defined to be the inverse of the rate.

4.3 Vector Commitment

A vector commitment scheme over some finite alphabet Σ is a tuple of algorithms $(\mathbf{Gen}, \mathbf{Digest}, \mathbf{Open}, \mathbf{Vf})$:

- $\mathsf{crs} \leftarrow \mathbf{Gen}(1^\lambda)$: on input the security parameter 1^λ, output a common reference string crs;
- $(\mathsf{cm}, \mathsf{aux}) \leftarrow \mathbf{Digest}(\mathsf{crs}, \mathsf{msg})$: given crs and a message $\mathsf{msg} \in \Sigma^\ell$, output a digest cm and some auxiliary information aux — we may assume that aux contains the message length $\ell := |\mathsf{msg}|$;
- $\pi \leftarrow \mathbf{Open}(\mathsf{crs}, \mathsf{aux}, Q)$: on input crs, auxiliary information aux (assumed to contain the message length ℓ), and a query set $Q \subseteq [\ell]$, output an opening proof π that $\mathsf{msg}[Q]$ is a restriction of msg to the indices Q;
- $(0,1) \leftarrow \mathbf{Vf}(\mathsf{crs}, \ell, \mathsf{cm}, Q, \mathsf{ans}, \pi)$: on input crs, message length ℓ, cm, a query set $Q \subseteq [\ell]$, a purported answer ans, and a proof π, outputs either 0 or 1 indicating reject or accept.

Additional assumption on the vector commitment. We shall assume that the opening proof π for the set Q of indices consists of an individual opening proof $\pi[q]$ for each $q \in Q$. Further, the verification algorithm $\mathbf{Vf}(\mathsf{crs}, \ell, \mathsf{cm}, Q, \mathsf{ans}, \pi)$ simply checks individually that for each $q \in Q$, $\pi[q]$ is a valid opening proof for $\mathsf{ans}[q]$ where $\mathsf{ans}[q]$ denotes answer to the query q contained in ans. Without risk of ambiguity, we use the notation $\mathbf{Vf}(\mathsf{crs}, \ell, \mathsf{cm}, q, \mathsf{ans}[q], \pi[q])$ to denote this individual check.

Correctness. Correctness requires that for any $\lambda \in \mathbb{N}$, any ℓ, any message $\mathsf{msg} \in \Sigma^\ell$, any $Q \subseteq [\ell]$, the following holds with probability 1: let $\mathsf{crs} \leftarrow \mathbf{Gen}(1^\lambda)$, $(\mathsf{cm}, \mathsf{aux}) \leftarrow \mathbf{Digest}(\mathsf{crs}, \mathsf{msg})$, $\pi \leftarrow \mathbf{Open}(\mathsf{crs}, \mathsf{aux}, Q)$, then it holds that $\mathbf{Vf}(\mathsf{crs}, \ell, \mathsf{cm}, Q, \mathsf{msg}[Q], \pi) = 1$.

Collision resistance. We say that a vector commitment scheme satisfies collision resistance (also called computationally binding in some literature) against size-$W(\cdot)$ adversaries, iff for any non-uniform probabilistic machine $\mathcal{A}(1^\lambda, \cdot)$ whose running time is bounded by $W(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the probability that the following experiment outputs 1 is at most $\mathsf{negl}(\lambda)$:

- $\mathsf{crs} \leftarrow \mathbf{Gen}(1^\lambda)$;
- $(\ell, \mathsf{cm}, \mathsf{ans}, \mathsf{ans}', Q, Q', \pi, \pi') \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs})$ where $Q, Q' \subseteq [\ell]$;
- Output 1 if $\mathbf{Vf}(\mathsf{crs}, \ell, \mathsf{cm}, Q, \mathsf{ans}, \pi) = \mathbf{Vf}(\mathsf{crs}, \ell, \mathsf{cm}, Q', \mathsf{ans}', \pi')$; however, there is some $i \in Q \cap Q'$ such that ans and ans' contain different answers for the index i.

Intuitively, collision resistance ensures that a computationally bounded adversary cannot open the same position to two different values.

Merkle [Mer89] showed how to build such a vector commitment scheme secure against polynomially sized adversaries (or quasi-polynomially sized adversaries resp.) assuming the existence of a collision resistant hash family secure against polynomially sized adversaries (or quasi-polynomially sized adversaries).

4.4 Random Strings are Incompressible

We will use the following generalization of Shannon's theorem.

Fact 4.1 (Extension of Shannon's theorem for codes with probabilistic correctness.)**.** *Suppose there is a randomized encoding procedure* $\mathsf{Enc} : \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^m$ *and decoding procedure* $\mathsf{Dec} : \{0,1\}^m \times \{0,1\}^r \to \{0,1\}^n$ *such that*

$$\Pr\left[r \xleftarrow{\$} \{0,1\}^r, \mathsf{msg} \xleftarrow{\$} \{0,1\}^n : \mathsf{Dec}\left(\mathsf{Enc}(\mathsf{msg}, r), r\right) = \mathsf{msg}\right] \geq \delta$$

Then, $m \geq n - \log(\frac{1}{\delta})$*.*

5 Construction for a Static Database

5.1 Intuition

We illustrate our static construction in Figure 1. Specifically, the database DB is erasure coded into $\overline{\mathsf{DB}}$, and ϕ^{DB} is the vector commitment (e.g., Merkle digest) of $\overline{\mathsf{DB}}$. A node *id* will sample a shard of $\overline{\mathsf{DB}}$ by computing $G(id)$, i.e., by hashing its identity using a random oracle $G(\cdot)$. It then computes a replication encoding (denoted $\mathbf{h}$) of its shard using the scheme of Pietrzak [Pie19]. Let ϕ^{shard} be the vector commitment of this replication encoding $\mathbf{h}$. In Figure 1, s denote the approximate space needed for each node to participate. The replication encoding has $4s$ vertices but the node only needs to store the last s vertices (henceforth also called the challenge set) plus a small amount of additional auxiliary data. Roughly speaking, the erasure coding is necessary for achieving best-possible recoverability — without it, the audit may fail to detect it with noticeable probability when a small number of blocks are missing globally. The replication encoding is necessary for preventing a storage node from reusing the same space to claim multiple rewards.

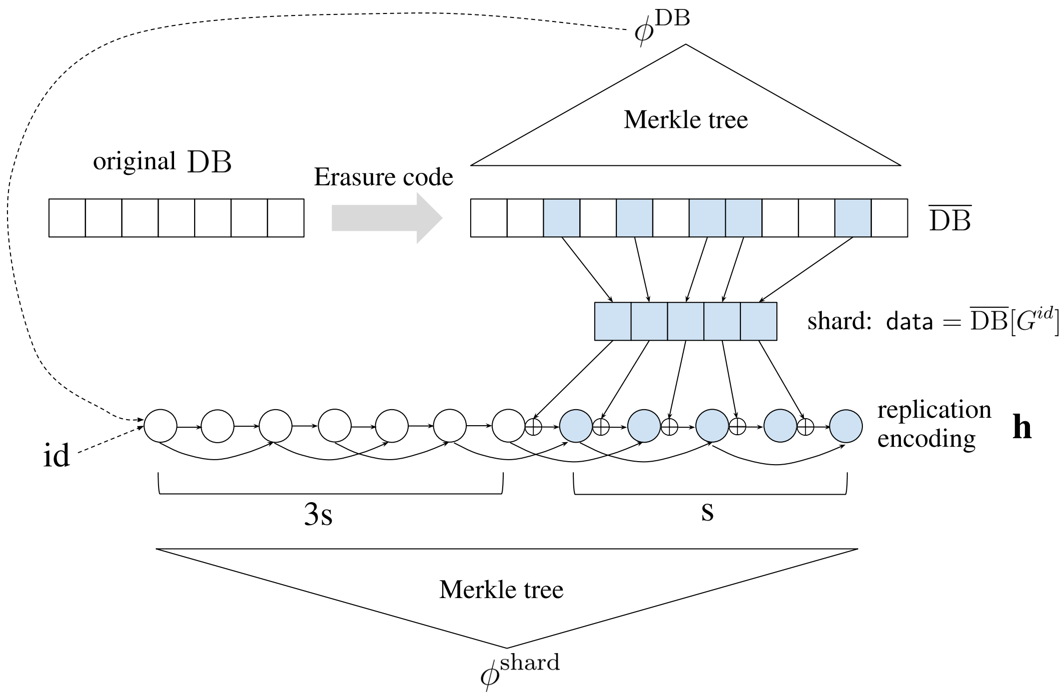

Figure 1: Warmup: construction for a static database.

The main technicality in the static construction arises from the fact that the storage node (i.e., prover) may be malicious. Pietrzak [Pie19]'s work focused on the honest prover case and did not give a full construction or proof for the malicious prover case. It turns out that there are some technicalities in extending their scheme to the malicious prover case. Specifically, we need a way for the node to prove correctness[3] of its purported commitment ϕ^{shard}. One naïve way to accomplish this is to rely on a generic SNARK. This is undesirable not only because of the extra computational costs, but also because this SNARK would have to prove computations that involve calls to a random oracle. Recent work [BCG24] has shown that constructing such a relativized SNARK is impossible. To avoid using a SNARK, the audit protocol is divided into an *offline* challenge and an *online* challenge. Specifically, the offline challenge proves that ϕ^{shard} is *approximately* correct, and the online challenge is essentially the challenge protocol of the original Pietrzak construction [Pie19].

- *Offline challenge.* The offline challenges are sampled by the node itself using the Fiat-Shamir paradigm, by computing a set of random indices $Q = \mathsf{FS}(\phi^{\text{shard}}, id)$ where $\mathsf{FS}(\cdot)$ denotes the Fiat-Shamir random oracle. Therefore, the node can prepare the response to the offline challenges prior to the audit protocol (hence the name "offline").

  The node then opens up 1) all positions corresponding to indices in Q in the replication encoding as well as all positions they depend on, denoted $\{\mathbf{h}[\mathsf{Nb}(q)]\}_{q \in Q}$ where

  $$\mathsf{Nb}(q) := \{q\} \cup \mathsf{parents}(q);$$

  and 2) all dependent data blocks in its shard denoted $\{\mathsf{data}[\mathsf{Nb}(q - 3s)]\}_{q \in Q'}$ where $Q' \subseteq Q$ denotes the indices in Q that are greater than $3s$. The auditor checks that all opened positions

[3] This is also the reason why Figure 1 opens up Pietrzak [Pie19]'s replication encoding — the internal workings of their construction are needed for the node to prove correctness of its replication encoding.

are consistent with the purported ϕ^{shard}, and that all positions in Q are computed correctly using the dependent positions in $\mathbf{h}$ as well as the dependent data blocks in the shard.

- *Online challenge.* The online challenges denoted $\widetilde{Q}$ are chosen at random by the challenger. Because in Pietrzak's replication encoding [Pie19], only the data-dependent indices $[3s+1, 4s]$ belong to the challenge set, the online challenges are sampled from $[3s+1, 4s]$, which is different from the offline challenges which are sampled from the entire range $[4s]$.

The careful reader may have noticed that the offline challenge falls short in proving that ϕ^{shard} is *strictly* correct. Instead, it only guarantees that ϕ^{shard} is a commitment of a replication encoding that is correct in most positions, also said to be *approximately* correct. The approximate correctness will bring some technicalities in our formal proofs later, but we show that nonetheless it is sufficient for establishing the desired security properties.

5.2 Formal Description

Notation and building blocks. We first define some notation and the underlying building blocks.

- Let $\mathsf{VC} = (\mathbf{Gen}, \mathbf{Digest}, \mathbf{Open}, \mathbf{Vf})$ denote a vector commitment scheme. In this paper, we assume that VC is instantiated with a Merkle tree using a hash function H_{vc}. When the hash function H_{vc} is modeled as a random oracle (RO), the scheme is collision resistant against any adversary that makes at most quasipolynomially many queries to H_{vc} as long as the output length of H_{vc} is $\omega(\log \lambda)$.
- Let $\mathsf{EC} = (\mathbf{Encode}, \mathbf{Decode})$ denote an erasure code with rate $1/R$ where R may be a function of the other parameters.
- Let n denote the size of the database assumed to be a power of 2, and let s denote the amount of space per node dedicated to storing the database.
- Let DRG be the depth-robust graph described by Alwen et al. [ABP18] with $4s$ vertices. We use the notation $\mathsf{parents}(i)$ to denote the parents of the i-th vertex in DRG, where $i \in [4s]$.
- Let $G : \{0,1\}^* \to [R \cdot n]^s$ denote a random oracle that samples the data to be stored by each node. On receiving some input from $\{0,1\}^*$, G outputs s randomly sampled indices from $[R \cdot n]$ — here, we assume sampling with replacement. For convenience, we sometimes use the notation $G(\mathsf{inp})[i]$ to mean the i-th index contained in the set $G(\mathsf{inp})$ where $i \in [s]$.
- Let $H : \{0,1\}^* \to \{0,1\}^B$ be a random oracle for constructing the replication code where B denotes the size of a block. For convenience, we often use the notation $H_\rho(\cdot)$ to mean the random oracle $H(\cdot)$ seeded with the string ρ, that is, $H_\rho(\mathsf{inp}) := H(\rho || \mathsf{inp})$.
- Let $\mathsf{FS} : \{0,1\}^* \to [4s]^\kappa$ be a random oracle for offline sampling challenges using the Fiat-Shamir paradigm.

Static construction. Our construction for a static database enjoys the same syntax as the definitions in Section 3, except that we do not need to support the **Update** function. Without loss of generality, we may assume that the size of the database DB is a power of 2.

- $\mathbf{Setup}(1^\lambda)$: let $\mathsf{crs} \leftarrow \mathsf{VC}.\mathbf{Gen}(1^\lambda)$ and output crs.
- $\mathbf{Init}(\mathsf{crs}, s, \mathsf{DB})$: Let DB be a database of size n.
    1. Compute the encoding $\overline{\mathsf{DB}} = \mathsf{EC}.\mathbf{Encode}(\mathsf{DB})$, and compute $(\phi^{\text{DB}}, \mathsf{aux}^{\text{DB}}) \leftarrow \mathsf{VC}.\mathbf{Digest}(\overline{\mathsf{DB}})$.

2. Output ϕ^{DB} as the public digest, and the publisher's state st^0 is set to be $(\phi^{\mathrm{DB}}, \mathsf{DB}, \overline{\mathsf{DB}}, \mathsf{aux}^{\mathrm{DB}})$.

Although not explicitly denoted, we may assume that the per-node space parameter s is saved along with ϕ^{DB}, and all algorithms below can access it.

- **Join**(st^0, id): The storage node then interacts with the publisher as follows — henceforth, let $G^{id} := G(id)$:

1. *Retrieve shard.* The publisher computes $\boldsymbol{\pi}^{\mathrm{DB}} \leftarrow \mathsf{VC}.\mathbf{Open}(\mathsf{crs}, \mathsf{aux}^{\mathrm{DB}}, G^{id})$, and sends ϕ^{DB}, $\mathsf{data} := \overline{\mathsf{DB}}[G^{id}]$, and the opening proofs $\boldsymbol{\pi}^{\mathrm{DB}}$ to the node.
2. *Compute replication code* $\mathbf{h}$. The storage node now computes a replication code over its shard as follows using $\rho = (\phi^{\mathrm{DB}}, id)$ as the seed. For $i = 1$ to $4s$, compute

$$\mathbf{h}[i] := \begin{cases} H^\rho(i, \mathbf{h}[\mathsf{parents}(i)]) & \text{if } i \leq 3s \\ H^\rho(i, \mathbf{h}[\mathsf{parents}(i)]) \oplus \mathsf{data}\,[i - 3s] & \text{o.w.} \end{cases}$$

In the above, if $\mathsf{parents}(i)$ output a set, then $\mathbf{h}[\mathsf{parents}(i)] := \{\mathbf{h}[j]\}_{j \in \mathsf{parents}(i)}$. Further, we define $\mathbf{h}[\mathsf{parents}(1)]$ to be the empty string, since the first vertex does not have any parents.
3. *Compute offline proof.* Below let $Q = \mathsf{FS}(\phi^{\mathrm{shard}}, id)$, and let $\mathsf{Nb}(q) = \{q\} \cup \mathsf{parents}(q)$.
   - Compute $(\phi^{\mathrm{shard}}, \mathsf{aux}^{\mathrm{shard}}) \leftarrow \mathsf{VC}.\mathbf{Digest}(\mathsf{crs}, \mathbf{h})$.
   - Compute $\boldsymbol{\pi}^{\mathrm{shard}} \leftarrow \mathsf{VC}.\mathbf{Open}(\mathsf{crs}, \mathsf{aux}^{\mathrm{shard}}, \{\mathsf{Nb}(q)\}_{q \in Q})$.
4. The node stores the following where $Q' \subseteq Q$ is the set of challenges greater than $3s$:

$$\mathsf{st}^{id} := \left(\mathbf{h}[3s+1 : 4s], \phi^{\mathrm{DB}}, \phi^{\mathrm{shard}}, \mathsf{aux}^{\mathrm{shard}}, \boldsymbol{\pi}^{\mathrm{DB}}, \boldsymbol{\pi}^{\mathrm{shard}}, \{\mathbf{h}[\mathsf{Nb}(q)]\}_{q \in Q}, \{\mathsf{data}[q - 3s]\}_{q \in Q'}\right)$$

The publisher's state st^0 is unchanged.

- **Audit**($(\mathsf{crs}, \phi^{\mathrm{DB}}, id), \mathsf{st}^{id}$): The auditor and the node interact as follows. Finally the auditor outputs 1 if all checks pass, and moreover, the node responds to the challenges within s rounds of time:

1. *Offline challenge.* Below, let $Q = \mathsf{FS}(\phi^{\mathrm{shard}}, id)$, and let $Q' \subseteq Q$ be the set of challenges greater than $3s$.
   - The storage node sends ϕ^{shard}, $\boldsymbol{\pi}^{\mathrm{DB}}$, $\boldsymbol{\pi}^{\mathrm{shard}}$, $\{\mathbf{h}[\mathsf{Nb}(q)]\}_{q \in Q}$, and $\{\mathsf{data}[q - 3s]\}_{q \in Q'}$ to the auditor.
   - The auditor verifies the following where we overload notation and let $\mathbf{h}[\mathsf{Nb}(q)]$ to mean the purported labels received by the auditor.

$$\mathsf{VC}.\mathbf{Vf}\left(\mathsf{crs}, 4s, \phi^{\mathrm{shard}}, \{\mathsf{Nb}(q)\}_{q \in Q}, \{\mathbf{h}[\mathsf{Nb}(q)]\}_{q \in Q}, \boldsymbol{\pi}^{\mathrm{shard}}\right) = 1$$

   - The auditor checks the following where we use the notation $\{\mathsf{data}[q - 3s]\}_{q \in Q'}$ to mean the corresponding terms received by the auditor:

$$\mathsf{VC}.\mathbf{Vf}\left(\mathsf{crs}, n, \phi^{\mathrm{DB}}, \{G^{id}[q - 3s]\}_{q \in Q'}, \{\mathsf{data}[q - 3s]\}_{q \in Q'}, \boldsymbol{\pi}^{\mathrm{DB}}\right) = 1$$

   - For each $q \in Q$, the auditor checks that

$$\mathbf{h}[q] = \begin{cases} H^\rho(q, \mathbf{h}[\mathsf{parents}(q)]) & \text{if } q \leq 3s \\ H^\rho(q, \mathbf{h}[\mathsf{parents}(q)]) \oplus \mathsf{data}[q - 3s] & \text{o.w.} \end{cases}$$

   where $\rho = (\phi^{\mathrm{DB}}, id)$.

2. *Online challenge.* The challenger samples a fresh online challenge set $\widetilde{Q} \xleftarrow{\$} [3s+1:4s]^\kappa$, and sends $\widetilde{Q}$ to the storage node. The storage node responds with $\mathbf{h}[\widetilde{Q}]$ as well as the opening proofs $\widetilde{\boldsymbol{\pi}}^{\text{shard}} \leftarrow \mathsf{VC}.\mathbf{Open}(\mathsf{crs}, \mathsf{aux}^{\text{shard}}, \{\mathsf{Nb}(q)\}_{q \in \widetilde{Q}})$. The verifier checks that

$$\mathsf{VC}.\mathbf{Vf}(\mathsf{crs}, 4s, \phi^{\text{shard}}, \{\mathsf{Nb}(q)\}_{q \in \widetilde{Q}}, \mathbf{h}[\widetilde{Q}], \widetilde{\boldsymbol{\pi}}^{\text{shard}}) = 1$$

where we overload the notation $\mathbf{h}[\widetilde{Q}]$ to mean the corresponding terms received by the auditor.

6 Construction for an Evolving Database

While the static construction itself (Section 5) is already nontrivial, our most novel contributions lie in extending this scheme to support a dynamically evolving database. What is most interesting here is that the common wisdom of "just applying the hierarchical data structure [BS80]" fails in multiple dimensions, as explained earlier in Section 2.2 and Section 2.3 — we suggest reading these sections first to get an intuition of the challenges before proceeding to our detailed scheme description below.

6.1 Basic Scheme

We will use the hierarchical data structure initially proposed by Bentley and Saxe [BS80] to make our scheme dynamic. However, there are a couple of important technicalities.

By variable renaming, we may assume that a per-node space requirement of S means that each node will be asked to store up to $(1 + o(1)) \cdot S \cdot B$ amount of data where B denotes the block size, and $o(1)$ is some appropriate sub-constant function in λ and the database size.

Data structure. We will maintain a hierarchical data structure of $L+1$ levels numbered $0, 1, \ldots, L$. Each level $\ell \in \{0, 1, \ldots, L\}$ is either empty or a static scheme denoted DDA_ℓ (described in Section 5) for data of size $n = 2^\ell$. All levels share the same crs, but each level has its own independent random oracle instances denoted G_ℓ, H_ℓ, and FS_ℓ. Moreover, we assume that $G_\ell(id)$ outputs s_ℓ randomly sampled indices.

Space allocation among levels. In our dynamic construction, we will periodically recompute the space allocation among the levels as the database grows. Suppose we need to reallocate the space at some point when the database has size n. Henceforth, we abuse notation and use $\omega(1)$ to denote an arbitrarily small super-constant function in λ. We define the following parameters:

- let $L(n) := \lfloor \log_2 n \rfloor$ be the largest level;
- let $\gamma(n) = S/n$;
- let $s_{\text{lb}}(n) = \gamma \cdot 2^L / (\omega(1) \cdot \log n)$ be the lower bound on the number of blocks to be sampled per level;
- for each non-empty level ℓ (determined by n), let $s_\ell(n) := \left\lceil \max(\gamma \cdot 2^\ell, s_{\text{lb}}) \right\rceil$ be the number of blocks a node will sub-sample for level ℓ.

In other words, we divide the levels into two categories:

1. **Large levels.** For the largest $\lfloor \log_2(\log n \cdot \omega(1)) \rfloor$ levels, the blocks are sampled at uniform density (modulo rounding errors) determined by the parameter γ. In other words, each node dedicates twice as much space for storing samples from level $i+1$ as from level i.

2. **Small levels.** For all remaining levels, a node samples the same number of blocks $\lceil s_{\mathrm{lb}} \rceil$ per level. In other words, each block in level i will be stored twice as often as a block in level $i+1$.

Fact 6.1. *Suppose $n \geq \lambda$. The above space allocation scheme has the following properties:*

- *The small levels account for $O(n/(\omega(1) \cdot \log n)) = n \cdot o(1)$ blocks in the database, and each node dedicates $O(S/\omega(1)) = S \cdot o(1)$ for storing samples from this portion of the database.*
- *The large levels account for $n \cdot (1 - \frac{1}{\omega(1)\cdot \log n}) = n \cdot (1 - o(1))$ blocks in the database, and each node dedicates $S \cdot (1 - \frac{1}{\omega(1)\cdot\log n}) = S \cdot (1 - o(1))$ blocks of local space for storing samples from this portion of the database.*

Further, the total number of blocks of local space allocated across all levels is at most $S(1 + o(1))$.

In the above, each $o(1)$ represents a (possibly different) sub-constant function in λ.

Parameter assumptions and choices. In our dynamic construction below, the parameter assumptions and choices are as follows where N denotes the maximum number of blocks.

- Number of challenges per level $\kappa = \omega(\log \lambda)$.
- Redundancy of the erasure code $R = e^{O(1)/\epsilon}$.
- $S = \omega(\kappa \cdot \log^2 \lambda)$, and we need each storage node to allocate $(1 + o(1))B \cdot S$ bits of space for some suitable sub-constant function $o(1)$ in λ.
- Recall that $s_{\mathrm{lb}} = \gamma \cdot 2^L/(\omega(1) \cdot \log n)$. Given $S = \omega(\kappa \cdot \log^2 \lambda)$, we can always choose the super-constant function $\omega(1)$ to be sufficiently small such that $s_{\mathrm{lb}} = \omega(\log \lambda)$.
- $B = \omega(\lambda_{\mathrm{vc}} \log N)$ where λ_{vc} denotes the hash output length of the VC scheme, assuming that VC is instantiated with a Merkle tree.
- The initial database size $n_0 \geq \max(S, \lambda)$.

Specifically, Theorem 8.12 and Theorem 9.4 require that $\kappa = \omega(\log \lambda)$, $R = e^{O(1)/\epsilon}$, $S = \omega(\log^2 \lambda)$, and $s_{\mathrm{lb}} = \omega(\log \lambda)$. Further, we need that $B = \omega(\lambda_{\mathrm{vc}} \log N)$ and $S = \omega(\kappa \log^2 \lambda)$. to ensure that all metadata stored by a node — including the Merkle proofs, the metadata needed to answer the offline challenge — occupies only $S \cdot o(1)$ amount of space.

Dynamic construction. We now describe our construction that works for an evolving database. Since we care about asymptotic behavior, without loss of generality, we may assume that the initial database size $n_0 \geq \lambda$.

- **Setup**(1^λ): call $\mathsf{crs} \leftarrow \mathsf{VC}.\mathbf{Gen}(1^\lambda)$ and output crs.
- **Init**$(\mathsf{crs}, S, \mathsf{DB})$: Let DB be the initial database containing n_0 blocks, and let $L = \lfloor \log_2 n_0 \rfloor$. Compute $s_\ell := s_\ell(n_0)$ for each non-empty level ℓ. Write $n_0 := \sum_{\ell \in \{0,1,\ldots,L\}} b_\ell \cdot 2^\ell$ where each $b_\ell \in \{0, 1\}$.
  - Divide DB into smaller databases of sizes $\{2^\ell : \forall \ell \text{ s.t. } b_\ell = 1\}$ each. Henceforth, let DB_ℓ denote the sub-database of size 2^ℓ.
  - For any ℓ such that b_ℓ is non-zero, call $(\phi_\ell, \mathsf{st}^0_\ell) \leftarrow \mathsf{DDA}_\ell.\mathbf{Init}(\mathsf{crs}, s_\ell, \mathsf{DB}_\ell)$. For all remaining levels ℓ where $b_\ell = 0$, let $\phi_\ell = \mathsf{st}^0_\ell = \bot$.
  - Output public digest $\boldsymbol{\phi} := \{\phi_\ell\}_{\ell \in \{0,\ldots,L\}}$, and publisher internal state $\mathsf{st}^0 := \{\mathsf{st}^0_\ell\}_{\ell \in \{0,\ldots,L\}}$.

Although not explicitly denoted, henceforth, we assume that the parameters γ and s_{lb} are saved along with the public digest $\boldsymbol{\phi}$ and can be accessed by all algorithms below.

- **Join**(st^0, id): Parse $\mathsf{st}^0 := \{\mathsf{st}^0_\ell\}_{\ell \in \{0,\ldots,L\}}$. For each non-empty level ℓ, the publisher and the node invoke $(\mathsf{st}^0_\ell, \mathsf{st}^{id}_\ell) \leftarrow \mathsf{DDA}_\ell.\mathbf{Join}(\mathsf{st}^0_\ell, id)$, except that in the computation of the replication code, for all levels, *we replace the seed with* $\rho := (\{\phi_0, \ldots, \phi_L\}, id)$.

  The publisher's new state is $\mathsf{st}^0 := \{\mathsf{st}^0_\ell\}_{\ell \in \{0,\ldots,L\}}$, and the node's new state is $\mathsf{st}^{id} := \{\mathsf{st}^{id}_\ell\}_{\ell \in \{0,\ldots,L\}}$.

- **Update**$((\mathsf{st}^0, \mathsf{upd}), \{\mathsf{st}^{id}\}_{id \in \mathsf{IDset}})$:

  1. Parse the state $\mathsf{st}^0 := \{\mathsf{st}^0_\ell\}_{\ell \in \{0,1,\ldots,L\}}$, where $\mathsf{st}^0_\ell := (\phi_\ell, \mathsf{DB}_\ell, \overline{\mathsf{DB}}_\ell, \mathsf{aux}^{\mathrm{DB}})$, and parse $\mathsf{st}^{id} := \{\mathsf{st}^{id}_\ell\}_{\ell \in \{0,\ldots,L\}}$.
  2. Let ℓ^* be the first empty level, and let $\mathsf{DB} = \mathsf{upd}||\mathsf{DB}_0|| \ldots ||\mathsf{DB}_{\ell^*-1}$.
  3. Let $(\phi_{\ell^*}, \mathsf{st}^0_{\ell^*}) \leftarrow \mathsf{DDA}_{\ell^*}.\mathbf{Init}(\mathsf{crs}, s_{\ell^*}, \mathsf{DB})$ where $s_{\ell^*} := \max(\lceil \gamma \cdot 2^{\ell^*} \rceil, s_{\text{lb}})$, and for all $\ell < \ell^*$, let $\phi_\ell = \mathsf{st}^0_\ell = \perp$.
  4. For each non-empty level ℓ, each node $id \in \mathsf{IDset}$ and the publisher invoke $(\mathsf{st}^0_\ell, \mathsf{st}^{id}_\ell) \leftarrow \mathsf{DDA}_\ell.\mathbf{Join}(\mathsf{st}^0_\ell, id)$, except that in the computation of the replication code[4], *replace the seed with* $\rho = (\{\phi_0, \ldots, \phi_L\}, id)$.
  5. Whenever the node's local space has exceeded $(1 + o(1)) \cdot n_{\text{prev}}$ where n_{prev} is the size of the database the last time the parameters were (re-)calculated, and $o(1)$ is a suitable sub-constant function, simply recalculate the parameters L and $\{s_\ell\}_\ell$ using the current n, and rerun the **Init** algorithm with the up-to-date database DB. Every node now reruns the **Join** algorithm with the publisher — see also Remark 3.
  6. Output the new public digest $\boldsymbol{\phi} := \{\phi_\ell\}_{\ell \in \{0,\ldots,L\}}$. The publisher's new state is $\mathsf{st}^0 := \{\mathsf{st}^0_\ell\}_{\ell \in \{0,\ldots,L\}}$, and the node's new state is $\mathsf{st}^{id} := \{\mathsf{st}^{id}_\ell\}_{\ell \in \{0,\ldots,L\}}$.

- **Audit**$((\mathsf{crs}, \boldsymbol{\phi}, id), \mathsf{st}^{id})$: Parse $\boldsymbol{\phi} := \{\phi_\ell\}_{\ell \in \{0,\ldots,L\}}$, and parse $\mathsf{st}^{id} := \{\mathsf{st}^{id}_\ell\}_{\ell \in \{0,\ldots,L\}}$. For each non-empty level ℓ in parallel, the node and the auditor invoke $\mathsf{DDA}_\ell.\mathbf{Audit}(\phi_\ell, \mathsf{st}^{id}_\ell)$, except that for all levels, 1) we *replace the seed with* $\rho = (\boldsymbol{\phi}, id)$; and 2) we require the answer to be sent within s_{lb} amount of time. The auditor outputs accept iff all instances output accept.

Remark 3 (Optimization for parameter refreshes)**.** When the level sizes $\{s_\ell\}_\ell$ are recomputed based on the new n, the level sizes can only decrease. We can use the same G_ℓ to decide which blocks to sample, and we will simply read the top s_ℓ indices sampled by the random oracle $G_\ell(id)$. Therefore, as an optimization, when the level sizes are recalculated, the node need not download any new block after s_ℓ shrinks; but it can drop some blocks it has already downloaded but are no longer needed.

6.2 Optimizations

So far, our scheme requires each node to expend roughly $S \cdot B$ bandwidth and computation per update. We now describe a few optimizations that get this cost down to $\widetilde{O}(1) \cdot B$. We stress that our update costs are nearly optimal. Specifically, it is inherent that each node must incur at least constant cost per update subject to our security definitions. For example, if 1% of the nodes do not learn any information about the new block, then an adversary selectively erase the remaining

[4]As an optimization, for any $\ell \neq \ell^*$, the node only needs to recompute its replication code locally using the new seed, and need not download its sub-sampled blocks again. See Section 6.2 for some additional optimizations.

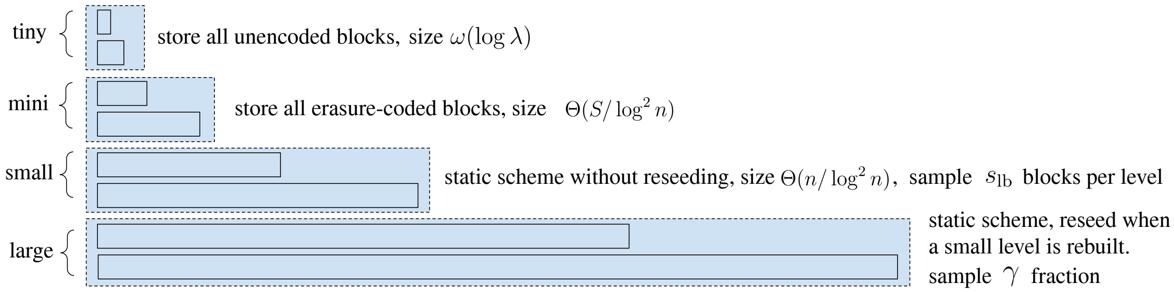

Figure 2: Final construction for a dynamic dataset

99% of the nodes, which will cause this block to be lost even if the combined space of the 1% of the nodes is sufficent for storing the whole dataset. After applying these optimizations, our final construction is shown in Figure 2.

Henceforth in our analysis, we use N to mean the maximum database size. Our cost analysis below will be amortized over $N - n_0$ number of updates, where n_0 is the initial database size.

Optimization 1: asymptotically improve per-node download. In our basic scheme, for every level ℓ, a node needs to download s_ℓ many erasure-coded blocks every 2^ℓ updates. This way, the amortized download bandwidth per node is at least s_{lb} which can be as large as $\frac{S}{\log N \cdot \omega(1)}$. To asymptotically improve the per-node download bandwidth, we can further refine our treatment of some of the smallest levels as below.

- *Tiny levels* where $2^\ell \leq \kappa$: Each node simply stores all 2^ℓ unencoded blocks belonging to the level, and the online challenge phase simply asks the node to open all of them.
- *Mini levels* where $2^\ell \in (\kappa, s_{\text{lb}}]$: We use an erasure code with redundancy $R = 2$ (or any constant $R > 1$) to encode the blocks in the level. Each node computes and stores a replication encoding over the *entire* set of erasure-coded blocks, i.e., the shard sampling using $G_\ell(id)$ is not needed.
- *Small levels where* $2^\ell \in (s_{\text{lb}}, \frac{s_{\text{lb}} \cdot n}{S}]$: treated in the same way as the small levels before.
- *Large levels*: All remaining levels, treated in the same way as the large levels before.

When n is growing over time, we can recompute the level definitions every time n grows by a factor of 2 — henceforth, the duration between the refreshes of level sizes is called a window.

We now account for the new download bandwidth under this improvement. It suffices to account for the cost of the most recent window amortized over the window itself. This is because costs over previous windows (amortized over the window itself) are dominated by the last window.

Every time there is an update, a storage node downloads the update itself. For every small or large level ℓ, every 2^ℓ updates, the node needs to download s_ℓ erasure-coded blocks. Without loss of generality, we may assume that s_{lb} is a power of 2. We may also assume that the choice of the super-constant function $\omega(1)$ is sufficiently small, such that the number of large levels is $O(\log \log \lambda)$. So the amortized number of blocks to download per update is

$$\sum_{\ell \in \text{ small } \cup \text{ large}} s_\ell / 2^\ell = 1 + \underbrace{1 + 1/2 + 1/4 + \ldots}_{\text{small levels}} + \underbrace{\frac{O(s_{\text{lb}})}{2^{\ell^*}} \cdot O(\log \log \lambda)}_{\text{large levels}} = O(1) + O(S \log \log \lambda / N)$$

where ℓ^* denotes the index of the smallest large level. Therefore, as long as $S = O(N / \log \log \lambda)$, we have that each node's amortized download bandwidth is $O(B)$.

It is not hard to see that the modifications to the tiny and mini levels do not affect our ϵ-best-recoverability and replication security guarantees (proven in Section 8 and Section 9). Specifically, for the tiny levels, ϵ-best-recoverability is trivial to prove. For the mini levels, the proof of ϵ-best-recoverability becomes simpler: since each node is required to store the entire erasure coded blocks, even extracting from just one node would suffice for the reconstruction of the level. For replication security, since the tiny and mini levels occupy only $o(1) \cdot S$, we can simply ignore the tiny and mini levels in the proof, and the $o(1)$ can be absorbed into the ϵ slack that we allow anyway, where ϵ is an arbitrarily small constant.

Optimization 2: asymptotically improve per-node computation. In our basic scheme of Section 6.1, every time an update comes in, every storage node must rebuild the replication encoding in all levels of the hierarchical data structure, thus incurring $O(B \cdot S)$ cost per update. We can further optimize the scheme to make each node's computation per update as small as $O(B \cdot \log N) \cdot \omega(1)$, where N is the maximum size of the database, and recall that $\omega(1)$ is the arbitrarily small super-constant function used in determining large and small levels. The idea is as follows:

- *Tiny level*: Use optimization 1.
- *Mini level*: Use optimization 1.
- *Small level*: For every small level ℓ, each node id uses the seed $\rho = (\phi_\ell, id)$ when computing the replication encoding, where ϕ_ℓ is the digest of the level ℓ itself. Only recompute the replication of the level when its seed changes.
- *Large level*: For every large level, each node id uses the seed $\rho = (\boldsymbol{\phi}^{\text{large}}, id)$ when computing the replication encoding, where $\boldsymbol{\phi}^{\text{large}} := \{\phi_\ell\}_{\ell \in \text{large}}$ denotes the digests of all large levels. Only recompute the replication of the level when its seed changes.

With this new variant, every small level ℓ only needs to refresh its replication encoding every 2^ℓ steps. Every large level must refresh its replication encoding every 2^{ℓ^*} times where ℓ^* denotes the index of the smallest large level. Therefore, the per-node per-update amortized computation cost is at most

$$B \cdot \left(\sum_{\ell \in \text{small}} \frac{1}{2^\ell} \cdot 2^\ell + \sum_{\ell \in \text{large}} \frac{1}{2^{\ell^*}} \cdot 2^\ell \right) \leq B \cdot \left(\underbrace{1 + 1 + \ldots + 1}_{O(\log N)} + 1 + 2 + 4 + \ldots + \omega(1) \cdot \log N \right)$$
$$\leq O(B \cdot \log N) \cdot \omega(1)$$

It is not hard to verify that the ϵ-best-recoverability and replication security proofs still hold with the above modification. Specifically, it suffices to apply the replication security proof to only the large levels, since the large levels occupy $1 - o(1)$ fraction of the node's local space S due to Fact 6.1. The proof of ϵ-best-recoverability is indifferent to what seed we use.

Efficiency. With these optimizations, we get the following efficiency under the parameter assumptions stated earlier in Section 6.1. Below, the costs are amortized over $N - n_0 = \mathsf{poly}(\lambda)$ number of updates.

- *Amortized per-node download bandwidth:* $B \cdot O(1 + S \log\log\lambda / N)$, which is simply $O(B)$ for the typical scenario when $S = O(N / \log\log\lambda)$.
- *Amortized per-node computation:* $O(B \cdot \log N) \cdot \omega(1)$ for an arbitrarily small super-constant function $\omega(1)$.

- *Amortized publisher computation.* Suppose we instantiate the erasure code with a special updatable erasure code such as the one proposed by Shi et al. [SSP13] — specifically, their erasure code provides an efficient update capability that allows us to accomplish level rebuild in time linear in the level's size. In this case, the amortized publisher computation for updating the erasure code is $O(B \cdot \log N) = e^{O(1)/\epsilon} \cdot \log N$, where the ϵ-dependent constant $e^{O(1)/\epsilon}$ comes from the redundancy of the erasure code. When $B = \omega(\lambda_{\text{vc}})$, the cost of updating the erasure-coded hierarchical data structure dominates the cost of recomputing the Merkle digests, so the total amortized publisher computation[5] is $B \cdot e^{O(1)/\epsilon} \cdot \log N$.
- *Audit cost.* The audit cost (including computation and communication) is at most $B \cdot \log \lambda \cdot \log N \cdot \omega(1)$, where the $\omega(1)$ term can be made an arbitrarily small super-constant function in λ. Specifically, under our assumption $B = \omega(\lambda_{\text{vc}} \cdot \log N)$, the cost of sending and verifying the VC opening proofs is absorbed by the cost of sending the challenged blocks.
- *Node space and join cost.* Each node's space is at most $B \cdot S \cdot (1 + o(1))$. To join the system, a node pays $O(S)$ cost including download bandwidth and local computation.

Remark 4 (Effect of periodic parameter refreshes)**.** Note that the need for a storage node to periodically refresh its level sizes $\{s_\ell\}_\ell$ based on the new n does not matter to the amortized costs. Due to Remark 3, the parameter refreshes do not increase a node's download costs. We now argue that the periodic refreshes do not affect the node's asymptotic computation cost. Recall that the parameter refresh happens only when $n_{\text{new}} \geq (1 + o(1)) \cdot n_{\text{prev}}$ for some suitable $o(1)$, where n_{new} is the current database size and n_{prev} is the database size when these parameters were last calculated. Thus, the $O(B \cdot S)$ cost for a storage node to recompute the replication code can be amortized to $o(1) \cdot n_{\text{prev}}$ steps. Therefore, assume that the initial database size $n_0 \geq S$, and we choose the sub-constant function $o(1)$ to be $\omega(1/\log \lambda)$, then the extra computational cost due to the refreshes is absorbed by the normal costs.

Remark 5 (Optimization for reducing working buffer needed)**.** Another small technicality is how much extra working buffer a node needs for computing the replication code. Recall that in level ℓ, the depth-robust graph has $4s_\ell$ vertices, but eventually the node stores only s_ℓ of them. We can make the amount of extra working buffer bounded by $B \cdot S \cdot o(1)$ with the following small modification: we restrict each replication code to be over at most $\nu(\lambda) \cdot S$ blocks for some sufficiently small super-constant function $\nu(\cdot)$. If a level ℓ has more than $\nu(\lambda) \cdot S$ blocks, we can just divide into multiple sub-levels each with at most $\nu(\lambda) \cdot S$ blocks when computing its replication code. This way, as long as the node computes the replication codes one after another, it only needs extra working buffer for one copy of the replication code, which is bounded by $B \cdot S \cdot o(1)$.

7 Extensions

7.1 Non-Uniform Node Space

So far, we have assumed a setting where all nodes have the same storage provisioning S. In practice, some nodes may be more powerful than others. The most naïve approach is for a node with $k \cdot S$ space to just spawn k instances. However, this approach would blow up the node's update and audit costs by a factor of k. We propose a simple modification to our scheme to avoid this k-factor

[5] The publisher's computation is essentially the same as in the dynamic PoR scheme of Shi et al. [SSP13], and using existing techniques [SSP13, BS80], the publisher's computation can be easily *deamortized*, where we spread the work evenly across all updates, thus avoiding some updates triggering a heavy-weight maintenance operation.

blowup. With our modification, a node with $k \cdot S$ space provisioning enjoys the same audit and update costs as a node with S space.

Observe that in our scheme, the publisher's data structure is agnostic of the parameter S. Therefore, instead of having global parameters S and $\{s_\ell\}_\ell$, we can make each node compute and maintain its own $\{s_\ell\}_\ell$ parameters based on its own space available. When the node builds its local replication encoding and computes the offline proofs, it samples the same number of challenges $\kappa = \omega(\log \lambda)$ as before, but the range from which the indices are sampled depends on $\{s_\ell\}_\ell$. The publisher need not know each node's local parameters, since during the **Join** and **Update** operations, the node simply downloads some portions of st^0 from the publisher. During the audit, the node can declare some purported space provisioning to the auditor upfront. The auditor then computes the $\{s_\ell\}_\ell$ parameters, and samples the challenges from the appropriate domains based on the $\{s_\ell\}_\ell$ parameters. Again, the number of challenges sampled remains unchanged, that is, $\kappa = \omega(\log \lambda)$.

It is not hard to see that our proofs still hold with this modification. Specifically, the replication security proofs (Section 8) hold directly even when all nodes have non-uniform space. For the approximate best-possible recoverability proofs (Section 9), the key observation is that if the adversary merges k adversarial identities into a single identity with k times the local space, then its advantage in Lemma 9.3 will only become smaller. One way to see this is that the effect of merging these identities is the same as querying the random oracle G on these identities separately, but when selecting a subset of μ identities, these merged identities must act as a bundle — either they are all selected or none of them are. With this key observation, it is easy to verify that the remainder of the proofs Section 9 hold even under non-uniform space.

7.2 Instantiating the Publisher Using IVC

If the scheme is used to back up blockchain data, then a trusted hash digest ϕ_{orig} of the original data is directly available from the consensus layer, and the blockchain will act as the auditor such that a node can get rewarded for its contributions. However, we need an additional publisher who must maintain an erasure-coded hierarchical data structure and the hash digests for this data structure. In practice, we can instantiate the publisher without having to trust the publisher, through the use of an Incrementally Verifiable Computation (IVC) scheme. Specifically, an untrusted publisher can compute the hash digests of the hierarchical data structure denoted $\boldsymbol{\phi} = (\phi_0, \ldots, \phi_L)$, and provide a succinct proof of correctness of $\boldsymbol{\phi}$ w.r.t. ϕ_{orig}.

Earlier works [DGKV22, PP22] have shown that under standard assumptions, we can construct an IVC scheme with the following efficiency: each update to a RAM machine incurs $\mathsf{poly}\log(\lambda, N)$ time to update the proof; and the proof size and verification time are also $\mathsf{poly}\log(\lambda, N)$, where N is the maximum number of RAM steps. When applied to our problem, the publisher can maintain the digests of the hierarchical data structure as well as the proofs of correctness in amortized time $\mathsf{poly}\log(\lambda, N)$ per update. As mentioned earlier, using existing techniques [SSP13, BS80], the prover's computation can also easily be deamortized over time. In a practical implementation, we can also use an IVC scheme based on non-falsifiable assumptions such as using recursive composition of SNARKs [BCCT13], or using more recent techniques [WPSP24].

8 Replication Security

8.1 Additional Preliminaries

Pebbling game. Let $\mathsf{DRG} = (V, E, V^C)$ be a directed acyclic graph, where V denotes the vertex set, E denotes the edge set, and $V^C \subseteq V$ denotes a subset of challenge vertices. The pebbling game on DRG played by an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is defined as follows:

- **Initialization:** $\mathcal{A}_1$ outputs an initial set of vertices $U \subseteq V$ to pebble.
- **Challenge:** Choose a random challenge $c \xleftarrow{\$} V^C$. $\mathcal{A}_2$ receives as input an initial set $U \subseteq V$ of vertices that have been pebbled, and the challenge c. $\mathcal{A}_2$ then proceeds in rounds, starting with round 1. In each round, $\mathcal{A}_2$ may pebble an arbitrary set of unpebbled vertices in DRG, subject to the constraint that a vertex can only be pebbled if all its parents are already pebbled in previous rounds. $\mathcal{A}$ is said to win the game if it successfully pebbles the vertex c.

We will use the depth-robust graph construction by Alwen et al. [ABP18]. Specifically, given an arbitrary $n \in \mathbb{N}$ and $\zeta > 0$, they define a depth-robust graph henceforth denoted $\mathsf{DRG}^{\zeta}_{4n}$ with $4n$ vertices, and prove that $\mathsf{DRG}^{\zeta}_{4n}$ is (e, d)-depth-robust for any $e + d \geq 1 - \zeta$.

Henceforth, an (s, t)-pebbling-adversary is one that is required to output at most s initial pebbles and answer the challenge in t rounds or fewer.

Lemma 8.1 (Pebbling hardness from depth-robust graphs [Pie19])**.** *Fix an arbitrary $n \in \mathbb{N}$ and $\zeta' > 0$. Consider the depth-robust graph $\mathsf{DRG}^{\zeta'}_{4n} = (V, E, V^C)$, where $V^C \subset V$ denotes the n topologically last vertices in V. Then, for any $\alpha \in [0, 1]$, for any deterministic (s, t)-adversary $\mathcal{A}$, the probability (over the choice of the challenge) that $\mathcal{A}$ wins the pebbling game on $\mathsf{DRG}^{\zeta'}_{4n}$ is at most ζ where*

$$s = n \cdot \alpha, \quad t = n, \quad \zeta = \alpha + 4\zeta'$$

8.2 Concurrent Pebbling Game

We consider an L-fold concurrent composition of the above pebbling game. Specifically, we now have L independent graph, and during the challenge phase, we issue one challenge per graph. The modified game is formally defined as follows.

L-fold concurrent pebbling. Let $\mathsf{DRG}_1, \ldots, \mathsf{DRG}_L$ be L DAGs (possibly of different sizes). The concurrent pebbling game on $\{\mathsf{DRG}_\ell\}_{\ell \in [L]}$ played by an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is defined as follows:

- **Initialization:** In each DAG, the adversary $\mathcal{A}_1$ specifies an initial set of vertices to pebble in each of the L graphs, henceforth denoted $U_1, \ldots, U_L$ respectively.
- **Challenge:** For $\ell \in [L]$, choose a random challenge $v^*_\ell \in \mathsf{ChSet}(\mathsf{DRG}_\ell)$ where we use the notation $\mathsf{ChSet}(\mathsf{DRG}_\ell)$ to denote the challenge set of vertices of DRG_ℓ. Now, $\mathcal{A}_2$ receives as input $\{U_\ell, v^*_\ell\}_{\ell \in [L]}$. $\mathcal{A}_2$ then proceeds in rounds, starting with round 1. In each round, $\mathcal{A}_2$ may pebble an arbitrary set of unpebbled vertices in $\{\mathsf{DRG}_\ell\}_{\ell \in [L]}$, subject to the constraint that a vertex can only be pebbled if all its parents are already pebbled in previous rounds. $\mathcal{A}$ is said to win iff for all $\ell \in [L]$, it has pebbled the v^*_ℓ-th vertex in DRG_ℓ.

Henceforth, an (s, t)-adversary is one who can output a total of at most s initial pebbles over all L graphs, and must respond to the challenge in t or fewer rounds.

Lemma 8.2 (L-fold concurrent pebbling hardness). *Fix arbitrary $n_1, \ldots, n_L \in \mathbb{N}$ and $\zeta' > 0$. For $\ell \in [L]$, suppose DRG_ℓ is the depth-robust graph $\mathsf{DRG}_{4n_\ell}^{\zeta'}$ of Alwen et al. [ABP18] with the challenge vertices being the topologically last n_ℓ vertices in its vertex set. Then, for any $\alpha \in [0,1]$, for any deterministic (s,t)-adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the pebbling game on $\{\mathsf{DRG}_\ell\}_{\ell\in[L]}$ is at most ζ where*

$$s = \alpha \cdot \sum_\ell n_\ell, \quad t = \min_{\ell\in[L]} n_\ell, \quad \zeta = (\alpha + 4\zeta')^L$$

Proof. Let $\alpha_\ell \cdot n_\ell$ be the number of initial pebbles placed on the j-th graph, where $\sum_{\ell\in[L]} \alpha_\ell \cdot n_\ell \leq \alpha \cdot \sum_\ell n_\ell$. Due to Lemma 8.2, for any fixed graph $\ell \in [L]$, the probability over the choice of v_ℓ^* that $\mathcal{A}$ successfully pebbles the v_ℓ^*-th vertex in the ℓ-th graph in $t = n_\ell$ rounds or fewer is at most $\alpha_\ell + 4\zeta'$. Recall that $\mathcal{A}$ only wins if it simulatenously pebbles the v_ℓ^*-th vertex in the ℓ-th graph for all $\ell \in [L]$. Therefore, the probability that $\mathcal{A}$ can win is upper bounded by $\prod_{\ell\in[L]}(\alpha_\ell + 4\zeta') \leq (\alpha + 4\zeta')^L$. □

8.3 The Underlying Pebbling Game of Our Construction

Our data archival scheme is associated with the following underlying pebbling game. Specifically, we will have μ independent instances, and each instance has L graphs. During the challenge phase, only one random instance will be challenged; moreover, for the selected instance, we will execute the challenge phase of the L-concurrent pebbling game κ independent times.

Formally, a $(\mu, \{n_\ell\}_{\ell\in[L]}, \kappa)$-pebbling game is defined as follows.

$(\mu, \{n_\ell\}_{\ell\in[L]}, \kappa)$-pebbling. Suppose we have μ instances, and each instance has L depth-robust graphs $\{\mathsf{DRG}_{4n_\ell}^{\zeta'}\}_{\ell\in[L]}$ of Alwen et al. [ABP18]. Again, for $\ell \in [L]$ the challenge vertices of $\mathsf{DRG}_{4n_\ell}^{\zeta'}$ are the topologically last n_ℓ vertices in its vertex set. The $(\mu, \{n_\ell\}_\ell, \kappa)$-pebbling game, played by an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, is defined as follows:

- **Initialization:** For each $i \in [\mu], \ell \in [L]$, $\mathcal{A}_1$ outputs an initial set of vertices $U_{i,\ell} \subseteq V_{i,\ell}$ to pebble.
- **Challenge:** Choose a random instance $i^* \stackrel{\$}{\leftarrow} [\mu]$. For κ times in parallel, perform the challenge phase of the L-concurrent pebbling game with $\mathcal{A}_2$ on the chosen instance. In other words, $\mathcal{A}_2$ is given the challenge instance i^*, and all the initial pebbled vertices $\{U_{i,\ell}\}_{i\in[\mu],\ell\in[L]}$. Now, repeat the following κ times *in parallel*:
  - Choose random challenge $v_\ell^* \stackrel{\$}{\leftarrow} \mathsf{ChSet}(\mathsf{DRG}_{4n_\ell}^{\zeta'})$ for each ℓ, and send $\{v_\ell^*\}_\ell$ to $\mathcal{A}_2$;
  - $\mathcal{A}_2$ now proceeds in rounds, starting with round 1. In each round, $\mathcal{A}_2$ may pebble an arbitrary set of unpebbled vertices in the L graphs of the i^*-th instance, subject to the constraint that a vertex can only be pebbled if all its parents are already pebbled in previous rounds.

  $\mathcal{A}$ is said to win the game iff for all κ parallel iterations, it simultaneously pebbles the challenged vertices in all L graphs in the challenge instance i^*.

Below, an $((s_1, \ldots, s_\mu), t)$-adversary is one who is required to output at most s_i initial pebbles for the i-th instance for any $i \in [\mu]$, and respond in t rounds or fewer.

Lemma 8.3. *Fix arbitrary $n_1, \ldots, n_L, \mu, \kappa \in \mathbb{N}$ and $\zeta' > 0$. Then, for any $\alpha_1, \ldots, \alpha_\mu \in [0,1]$, for any deterministic $((s_1, \ldots, s_\mu), t)$-adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, $\mathcal{A}$ can win the above $(\mu, \{n_\ell\}_\ell, \kappa)$-pebbling game with probability at most ζ where*

$$\forall i \in [\mu] : s_i = \alpha_i \cdot \sum_{\ell\in[L]} n_\ell, \qquad t = \min_{\ell\in[L]} n_\ell, \quad \zeta = \frac{1}{\mu} \cdot \sum_{i\in[\mu]} (\alpha_i + 4\zeta')^\kappa$$

Proof. Follows in a straightforward fashion from Lemma 8.2. Specifically, we may assume $\kappa = 1$ since for larger κ, $\mathcal{A}$'s winning probability cannot become be better. $\square$

8.4 Replication Security: Proofs

8.4.1 Encoding Algorithm

Suppose VC is instantiated using a Merkle tree where the hash function H_{vc} is modeled as a random oracle.

Given a deterministic machine $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, let $\mathcal{A}_1$ be the adversary that interacts with the challenger during the **Initialization** and **Queries** phases of the $\mathsf{PoRepExpt}$. At the end, $\mathcal{A}_1$ outputs μ challenge identities $id_1, \ldots, id_\mu$, as well as some internal state $\mathsf{st}^{\mathcal{A}}$ to be passed to $\mathcal{A}_2$. Next, we execute $\mathcal{A}_2$ who interacts with the challenger for the **Challenge** phase. Both $\mathcal{A}_1$ and $\mathcal{A}_2$ can make random oracle queries — henceforth, let P_1 and P_2 denote the maximum number of random oracle queries made by $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively.

For $\ell \in \{0, 1, \ldots, L\}$, let G_ℓ, H_ℓ, and FS_ℓ denote the random oracle instances for level ℓ. Let H_{vc} denote the global random oracle instance for the VC scheme.

The parallel adversary. Let $id_1, \ldots, id_\mu$ be the challenge identities output by $\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})$. Henceforth, we use the notation $\mathcal{B}^{id}(\mathsf{st}^{\mathcal{A}})$ where $id \in \{id_1, \ldots, id_\mu\}$ to denote the following algorithm that computes the responses to all possible online challenges in parallel:

- Execute $\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})$ until it outputs all the challenge identities, and let $\mathcal{A}_3^{id}(\mathsf{st})$ be its continuation upon receiving the challenge id where st denotes the internal state passed to $\mathcal{A}_3$.
- Execute $\mathcal{A}_3^{id}(\mathsf{st})$ until it finishes the offline challenge step. Let $\mathcal{A}_4^{id}(\mathsf{st}')$ be a continuation of $\mathcal{A}_3^{id}$ at this point where st' is the internal state passed to $\mathcal{A}_4^{id}$.
- Fork $\prod_\ell s_\ell^\kappa$ instances of $\mathcal{A}_4^{id}(\mathsf{st}')$ and execute them in parallel on all possible online challenges.

We devise the following encoding scheme $(\mathsf{Enc}, \mathsf{Dec})$:

Encoding algorithm $\mathsf{Enc}(\mathsf{msg}, r)$. We describe how to encode a message msg using the random coins r.

1. Using part of the random coins in r, guess at random which is the time step t^* in which the adversary $\mathcal{A}_1$ will for the first time make a query to either $H_\ell(\boldsymbol{\phi}^*, \cdot)$ for some ℓ, where $\boldsymbol{\phi}^* := (\phi_0^*, \ldots, \phi_L^*)$ denotes the digests of all levels when the challenge phase is invoked.

2. Execute $\mathcal{A}_1$ till t^*, and answer $\mathcal{A}_1$'s random oracle queries using the coins contained in r. If $\mathcal{A}_1$ does not make a query of the form $H_\ell(\boldsymbol{\phi}^*, \cdot)$ at time t^*, or this is not the first time a query of the form $H_\ell(\boldsymbol{\phi}^*, \cdot)$ has appeared for the observed choice of ϕ^*, simply abort and output $\bot$.

3. Program the random oracles $\{H_\ell(\boldsymbol{\phi}^*, \cdot)\}_{\ell \in \{0,\ldots,L\}}$ with msg, and for all other random oracle queries, answer using the coins contained in r.

   Continue executing $\mathcal{A}_1$ until it is about to enter the challenge phase. Let $\mathsf{st}^{\mathcal{A}}$ be its internal state. Let DB^* be the cumulative database at the challenge time. If the digest of DB^* does not agree with the guessed $\boldsymbol{\phi}^*$, then simply abort and output $\bot$.

4. Let $id_1, \ldots, id_\mu$ be the challenge identities output by $\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})$. Execute $\mathcal{B}^{id}(\mathsf{st}^{\mathcal{A}})$ until it finishes the offline challenge phase.

5. For each challenge identity id such that the adversary passes the offline challenge phase when challenged on id, for each non-empty level ℓ, do the following: let ϕ_ℓ^{shard} be the shard commitment submitted by $\mathcal{B}^{id}(\mathsf{st}^{\mathcal{A}})$ during the offline challenge phase and run the following extractor algorithm.

   Repeat the following T^{ext} times where $T^{\text{ext}} := T^{\text{ext}}(\lambda)$ is a super-polynomial function in λ:

   - Rewind the entire adversary $\mathcal{A}$ to the point when it first made a query to $\mathsf{FS}_\ell(\phi_\ell^{\text{shard}}, id)$. Execute $\mathcal{A}$ but from this point on, reprogram the answer to any fresh queries to $\mathsf{FS}_\ell(\cdot)$ where fresh means that the $\mathcal{A}$ has not made this query yet upto the rewinded point, including the query to $\mathsf{FS}_\ell(\phi_\ell^{\text{shard}}, id)$ itself. Here, we assume that the encoding algorithm will use its own internal randomness to reprogram the answers to $\mathsf{FS}_\ell(\cdot)$ queries. We can view internal random coins as a part of r that will not be consumed by the decoding algorithm.
   - If in this reprogrammed execution trace, the encoding algorithm has not aborted, $\mathcal{A}$ still includes id in the set of challenge identities, and $\mathcal{B}^{id}$ still passes the offline phase using the same shard commitment ϕ_ℓ^{shard}, then let $\{\mathbf{h}_\ell[\mathsf{Nb}(q)], \ldots\}_{q \in Q}$ be the answer returned by $\mathcal{B}^{id}(\mathsf{st}^{\mathcal{A}})$ during the offline challenge phase for level ℓ. Record the answers returned by the adversary by setting $\mathbf{h}_\ell^{id}[\mathsf{Nb}(q)] := \mathbf{h}_\ell[\mathsf{Nb}(q)]$ for each $q \in Q$ — initially, all entries of $\mathbf{h}_\ell^{id}$ were set to $\perp$.

   During the execution, if we ever observe a collision of the VC scheme, then abort and output $\perp$. Collisions include the following types: 1) we observe two different values $\mathbf{h}_\ell[q]$ and $\mathbf{h}'_\ell[q]$ that both have valid opening proofs w.r.t. ϕ_ℓ^{shard}; and 2) for some challenged index $q > 3s_\ell$, the adversary answers with the purported data entry $\mathsf{data}[q - 3s_\ell]$, however, $\mathsf{data}[q - 3s_\ell] \neq \overline{\mathsf{DB}}_\ell[G_{id}[q - 3s]]$.

6. For each $q \in Q$ and each non-empty ℓ, we say that the label $\mathbf{h}_\ell^{id}[q]$ is *correct* iff for any $p \in \mathsf{parents}(q)$, $\mathbf{h}_\ell^{id}[p] \neq \perp$, and moreover, $\mathbf{h}_\ell^{id}[q]$ is computed correctly from $\mathbf{h}_\ell^{id}[\mathsf{parents}(q)]$, as well as $\overline{\mathsf{DB}}_\ell[G_{id}[q - 3s_\ell]]$ if $q > 3s_\ell$.

7. For each $id \in \{id_1, \ldots, id_\mu\}$: execute $\mathcal{B}^{id}(\mathsf{st}^{\mathcal{A}})$ on online challenges concurrently. We say that the label $\mathbf{h}_\ell^{id}[q]$ is *predicted* iff $\mathbf{h}_\ell^{id}[q]$ is correct, and moreover, $\mathcal{B}^{id}(\mathsf{st}^{\mathcal{A}})$ submitted $\mathbf{h}_\ell^{id}[q]$ either in a random oracle query or when responding to some challenge, before it queried $H_\ell(\phi^*, id, q, \mathbf{h}_\ell^{id}[\mathsf{parents}(q)])$. The first random oracle query made by $\mathcal{B}^{id}(\mathsf{st}^{\mathcal{A}})$ that includes $\mathbf{h}_\ell^{id}[q]$ as input is said to be a *predicting* query.

8. Let $\eta' \in (0,1)$ be some suitably small constant. If the number of predicted labels is fewer than $(\overline{\alpha} - \eta') \cdot \mu \cdot \sum_\ell s_\ell$, output $\perp$, where $\overline{\alpha}$ denotes the expected fraction of challenges $(id, \{Q_\ell\}_\ell)$ that $\mathcal{A}$ can respond to under a random choice of msg and r conditioned on the encoding algorithm not aborting before entering the offline challenge phase.

   Otherwise, output the encoded message $\overline{\mathsf{msg}}$ consisting of the following terms:

   - $\mathsf{st}^{\mathcal{A}}$: the internal state output by $\mathcal{A}_1$;
   - pred: contains the following information for each predicting query: 1) the index of id within $\{id_1, \ldots, id_\mu\}$ pertaining to the predicting query, 2) the index of the predicting query among all queries made by $\mathcal{B}^{id}(\mathsf{st}^{\mathcal{A}})$ and 3) the starting offset of the input of the predicted label;
   - other: contains all other parts of msg that is not the answer to a predicting query.

   Note that if the number of predicted labels is greater than $(\overline{\alpha} - \eta) \cdot \mu \cdot \sum_\ell s_\ell$, we can simply truncate it to exactly $(\overline{\alpha} - \eta) \cdot \mu \cdot \sum_\ell s_\ell$ labels. This way, if the encoding is successful, then the encoded string always has a fixed length.

Decoding algorithm $\mathsf{Dec}(\overline{\mathsf{msg}}, r)$**.** If $\overline{\mathsf{msg}} = \bot$, output $\bot$, else parse $\overline{\mathsf{msg}} = (\mathsf{st}^{\mathcal{A}}, \mathsf{pred}, \mathsf{other})$ and continue as follows.

1. Using the same coins r, we execute $\mathcal{A}_1$ until the t^*-th random oracle query which is of the form $H_\ell(\boldsymbol{\phi}^*)$. Now, just like Step 2 of the Enc algorithm, we can use all the H_{vc} queries made so far to extract a DB^* that match $\boldsymbol{\phi}^*$. Let $\{\overline{\mathsf{DB}}_\ell\}_\ell$ be the erasure-coded levels derived from DB^*.
2. Let $id_1, \ldots, id_\mu$ be the challenge identities output by $\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})$. Execute Step 4 of the Enc algorithm using $\mathcal{B}^{id}(\mathsf{st}^{\mathcal{A}})$, except with the following modifications. Whenever $\mathcal{B}^{id}(\mathsf{st}^{\mathcal{A}})$ makes a random oracle query of the form $H_\ell(\phi^*, id, \cdot)$, use the information in pred to check to see if the query's answer can be reconstructed from some predicted label and $\overline{\mathsf{DB}}_\ell$. If so, use the appropriate predicted label (which must have been seen by now) and $\overline{\mathsf{DB}}_\ell$ to compute the answer. Otherwise, respond using other.
3. When the previous step finishes, we will have reconstructed all the predicted labels that are not recorded in other, and therefore we can recover msg.

The following simple fact holds by construction.

Fact 8.4. *If the encoding scheme does not output $\bot$, then the decoded result must be correct.*

8.4.2 Analysis of the Extractor

Suppose $\mathcal{A}$ makes at most N_{fs} queries to the random oracle FS. We define the following events:

- $\mathsf{Pass}_{\ell,j}$: Let $(\phi_\ell^{\mathrm{shard}}, id)$ be the pair $\mathcal{A}$ submits in the j-th query to FS. $\mathsf{Pass}_{\ell,j}$ is the event that the encoding algorithm has not aborted at the beginning of the offline challenge phase, id is the first challenge identity submitted by $\mathcal{A}$, and when challenged with id, $\mathcal{A}$ submits $\phi_\ell^{\mathrm{shard}}$ for level ℓ during the offline challenge phase, and responds correctly for level ℓ.
- Pass_ℓ: The encoding algorithm has not aborted at the beginning of the offline challenge phase, and moreover, when challenged on id_1, $\mathcal{A}$ responds correctly for level ℓ during the offline challenge phase, where id_1 is the first challenge identity submitted by $\mathcal{A}$.
- $\mathsf{ReconstrGood}_\ell$: The encoding algorithm reconstructs an array of labels $\mathbf{h}_\ell^{id_1}$ for level ℓ such that at least $1-\eta$ fraction of positions are correct, where id_1 is the first challenge identity submitted by $\mathcal{A}$, and $\eta \in (0,1)$ is a suitably small constant.
- Col: the event that some collision is observed when running the extractor;
- $\mathsf{Deficient}_\ell$: Let id_1 be the first challenge identity submitted by $\mathcal{A}$, and let $\phi_\ell^{\mathrm{shard}}$ be the shard commitment for level ℓ submitted by $\mathcal{A}$ when challenged on id_1. There exists some $q \in \mathsf{FS}(\phi_\ell^{\mathrm{shard}}, id_1)$ that in the reconstructed $\mathbf{h}_\ell^{id_1}$, the position q is not correct.

Although the above events focus on the first challenge identity output by the adversary, all lemmas proven below trivially extend when "first" is replaced with "any fixed i-th" challenge identity where $i \in [\mu]$.

Lemma 8.5 (Technical lemma for the extractor)**.** *For any fixed non-empty level ℓ,* $\Pr[\mathsf{Pass}_\ell \wedge \neg\mathsf{ReconstrGood}_\ell] \le \mathsf{negl}(\lambda)$. *As a direct corollary,* $\Pr[\forall$ *non-empty* $\ell : \mathsf{Pass}_\ell \wedge \neg\mathsf{ReconstrGood}_\ell] \le \mathsf{negl}(\lambda)$.

Proof of Lemma 8.5. Below we prove Lemma 8.5.

Claim 8.6. *Fix* ℓ *and* j*.* $\Pr[\mathsf{Pass}_{\ell,j} \wedge \mathsf{Col}] \leq \mathsf{negl}(\lambda)$*.*

Proof. Suppose that H_{vc} is a random oracle. Since $\mathcal{A}$ is polynomially bounded, the extractor makes at most $\mathsf{poly}(\lambda) \cdot T^{\mathrm{ext}}$ queries to H_{vc}. The probability that the extractor can find a collision in H_{vc} is negligibly small, as long as the output length of H_{vc} is at least $2.1 \log_2 T^{\mathrm{ext}}$ for a super-polynomial T^{ext}. □

Claim 8.7. *Fix* ℓ *and* j*.* $\Pr[\mathsf{Pass}_{\ell,j} \wedge (\mathsf{Col} \vee \mathsf{Deficient}_\ell)] \leq \mathsf{negl}(\lambda)$*.*

Proof. Observe that $\Pr[\mathsf{Pass}_{\ell,j} \wedge (\mathsf{Col} \vee \mathsf{Deficient}_\ell)] \leq \Pr[\mathsf{Pass}_{\ell,j} \wedge \mathsf{Col}] + \Pr[\mathsf{Pass}_{\ell,j} \wedge \neg\mathsf{Col} \wedge \mathsf{Deficient}_\ell]$. Due to Claim 8.6, it suffices to prove that $\Pr[\mathsf{Pass}_{\ell,j} \wedge \neg\mathsf{Col} \wedge \mathsf{Deficient}_\ell] \leq \mathsf{negl}(\lambda)$.

Henceforth in the proof, we may fix the all other random coins except the coins used in FS and used by the extractor, and the probabilities below are taken over the choice of FS and the coins consumed by the extractor. Let pre be the answers to the first $j-1$ queries to FS.

$$
\begin{aligned}
&\Pr[\mathsf{Pass}_{\ell,j} \wedge \neg\mathsf{Col} \wedge \mathsf{Deficient}_\ell] \\
\leq &\sum_{\mathsf{pre}} \Pr[\mathsf{Pass}_{\ell,j} \wedge \neg\mathsf{Col} \wedge \mathsf{Deficient}_\ell | \mathsf{pre}] \cdot \Pr[\mathsf{pre}] \\
\leq &\sum_{\mathsf{pre}} \sum_{q \in [4s_\ell]} \delta_{\ell,j,\mathsf{pre}}(q) \cdot (1 - \delta_{\ell,j,\mathsf{pre}}(q))^{T^{\mathrm{ext}}} \cdot \Pr[\mathsf{pre}] \\
\leq &\sum_{\mathsf{pre}} \sum_{q \in [4s_\ell]} \frac{1}{T^{\mathrm{ext}}} \cdot \Pr[\mathsf{pre}] \leq 4s_\ell / T^{\mathrm{ext}}
\end{aligned}
$$

where $\delta_{\ell,j,\mathsf{pre}}(q)$ is the probability that conditioned on pre, the encoding algorithm has not aborted when entering the offline challenge phase, q is included in the answer to the j-th FS query henceforth denoted $(\phi_\ell^{\mathrm{shard}}, id)$, id is the first challenge identity, and when challenged with id, the adversary submits $\phi_\ell^{\mathrm{shard}}$ in the offline challenge phase, and answers the challenge q correctly for level ℓ. The second inequality used the fact that $\delta \cdot (1-\delta)^T \leq 1/T$ for any $\delta \in (0,1)$. The above probability is negligibly small since s_ℓ is polynomially bounded in λ and T^{ext} is super-polynomial in λ.

□

Claim 8.8. *Fix* ℓ *and* j*.* $\Pr[\mathsf{Pass}_{\ell,j} \wedge \neg\mathsf{ReconstrGood}_\ell \wedge \neg\mathsf{Deficient}_\ell] \leq \mathsf{negl}(\lambda)$*.*

Proof. Observe that

$$
\begin{aligned}
\Pr[\mathsf{Pass}_{\ell,j} \wedge \neg\mathsf{ReconstrGood}_\ell \wedge \neg\mathsf{Deficient}_\ell] &\leq \Pr[\neg\mathsf{ReconstrGood}_\ell \wedge \neg\mathsf{Deficient}_\ell] \\
&\leq \Pr[\neg\mathsf{Deficient}_\ell | \neg\mathsf{ReconstrGood}_\ell]
\end{aligned}
$$

Since the coins in selecting the challenge $\mathsf{FS}(\phi_\ell^{\mathrm{shard}}, id)$ in the main execution path are independent of the coins consumed by the extractor, we conclude that $\Pr[\neg\mathsf{Deficient}_\ell | \neg\mathsf{ReconstrGood}_\ell] \leq (1-\eta)^\kappa$, which is negligibly small in λ for any fixed constant $\eta \in (0,1)$ and $\kappa = \omega(\log \lambda)$. □

Now, for a fixed ℓ and j, we have that

$$
\begin{aligned}
&\Pr[\mathsf{Pass}_{\ell,j} \wedge \neg\mathsf{ReconstrGood}_\ell] \\
\leq &\Pr[\mathsf{Pass}_{\ell,j} \wedge \neg\mathsf{ReconstrGood}_\ell \wedge (\mathsf{Col} \vee \mathsf{Deficient}_\ell)] + \Pr[\mathsf{Pass}_{\ell,j} \wedge \neg\mathsf{ReconstrGood}_\ell \wedge \neg\mathsf{Col} \wedge \neg\mathsf{Deficient}_\ell] \\
\leq &\Pr[\mathsf{Pass}_{\ell,j} \wedge (\mathsf{Col} \vee \mathsf{Deficient}_\ell)] + \Pr[\mathsf{Pass}_{\ell,j} \wedge \neg\mathsf{ReconstrGood}_\ell \wedge \neg\mathsf{Deficient}_\ell] \\
\leq &\mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda) \leq \mathsf{negl}(\lambda)
\end{aligned}
$$

Finally, we arrive at Lemma 8.5 by taking a union bound on j and observing the fact that except with negligible probability, if Pass_ℓ happens, then $(\phi_\ell^{\text{shard}}, id)$ must have been submitted in a query to FS prior to the adversary sends back the response in the offline challnege phase, where id is the first challenge identity, and ϕ_ℓ^{shard} is the shard commitment submitted by the adversary for level ℓ when challenged on id.

8.4.3 Probability of Successful Encoding and Decoding

During the Enc algorithm, we say that some challenge id is *good*, iff the adversary passes the offline challenge phase when challenged on id. Henceforth, let P_{id} be the number of predicted labels for the challenge identity id.

Claim 8.9. *For any fixed* msg *and* r*, the following holds. For each challenge identity* id*, let* $P'_{id} = P_{id} + 4\eta \cdot \sum_\ell s_\ell$ *if* id *is good, and else let* $P'_{id} = 0$*. Suppose that under the coins* r *and the message* msg*,* $\mathcal{A}$ *can pass the audit on* $\alpha = \alpha_{r,\mathsf{msg}}$ *fraction of the challenges* $(id, \{\widetilde{Q}_\ell\}_\ell)$ *where* $\widetilde{Q}_\ell$ *denotes the online challenge set for level* ℓ*. Then, there is some* $(\{P'_{id}\}_{id \in \{id_1,\ldots,id_\mu\}}, t)$*-adversary who can win the* $(\mu, \{s_\ell\}_\ell, \kappa)$*-pebbling game over at least* α *fraction of the challenges of the form* $(id, \{\widetilde{Q}_\ell\}_\ell)$ *where* $t = s_{\text{lb}}$*.*

Proof. Due to Lemma 8.5, if some challenge id is good, then for all non-empty ℓ, the extracted $\mathbf{h}_\ell^{id}$ array has at least $1 - \eta$ fraction of positions that are correct. In this case, we will place an initial pebble on a vertex if the vertex is associated with either a predicted or incorrect label. The total number of initial pebbles for instance id is upper bounded by $P'_{id} = P_{id} + \eta \cdot 4 \sum_\ell s_\ell$. We place no pebbles for any id that is not good since the adversary cannot even pass the offline challenge phase for a bad id.

Consider a good id. For any vertex indexed by (id, ℓ, q) that is not initially pebbled, if the adversary $\mathcal{B}^{id}$ submits the label $\mathbf{h}_\ell^{id}[q]$ during some random oracle query or in response to some challenge, it must be that the adversary $\mathcal{B}^{id}$ has queried $H_\ell(\boldsymbol{\phi}^*, id, q, \mathbf{h}_\ell^{id}[\mathsf{parents}(q)])$. If we order these queries in the order of q for each ℓ, it will give a way to pebble the level-ℓ graph of instance id. Further, if $\mathcal{B}^{id}$ succeeds in answering the online challenge $\{\widetilde{Q}_\ell\}_\ell$, then for every ℓ, every $q \in \widetilde{Q}_\ell$, the q-th vertex of the level-ℓ graph will have been pebbled at the end.

Therefore, with the above placed initial pebbles, the adversary can succeed in answering at least α fraction of the challenges. ☐

Claim 8.10. *Given any* msg *and* r*. If the encoding algorithm does not abort before entering the challenge phase, then* $\sum_{id \in \{id_1,\ldots,id_\mu\}} P_{id} \geq (\alpha - 4\zeta' - \eta) \cdot \mu \cdot \sum_\ell s_\ell$*, where* $\alpha := \alpha_{r,\mathsf{msg}}$ *is defined in the same way as in Claim 8.9.*

Proof. Henceforth, let $\alpha'_{id} := \frac{P'_{id}}{\sum_\ell s_\ell}$, and let $\alpha_{id} := \frac{P_{id}}{\sum_\ell s_\ell}$. Let good be the set of good identities. Due to Lemma 8.3, we have that

$$\sum_{id \in \mathsf{good}} (\alpha'_{id} + 4\zeta')^\kappa \geq \alpha \cdot \mu$$

Let $\mathsf{large} \subseteq \mathsf{good}$ be the set of identities such that for $id \in \mathsf{large}$, $\alpha'_{id} \geq 1 - 5\zeta'$. Let $\mathsf{small} =$

$\mathsf{good}\backslash\mathsf{large}$. We have that

$$\begin{aligned} &\sum_{id \in \{id_1,\ldots,id_\mu\}} (\alpha'_{id} + 4\zeta')^\kappa \\ \leq &\sum_{id \in \mathsf{large}} (\alpha'_{id} + 4\zeta')^\kappa + \sum_{id \in \mathsf{small}} (1 - \zeta')^\kappa \\ \leq &\sum_{id \in \mathsf{large}} (\alpha'_{id} + 4\zeta')^\kappa + \mathsf{negl}(\lambda) \\ \leq &\sum_{id \in \mathsf{large}} (\alpha'_{id} + 4\zeta') + \mathsf{negl}(\lambda) \end{aligned}$$

Thus, we have that

$$\sum_{id \in \mathsf{large}} \alpha'_{id} \geq (\alpha - 4\zeta') \cdot \mu \qquad (\star)$$

We also have

$$\sum_{id \in \{id_1,\ldots,id_\mu\}} \alpha_{id} \geq \sum_{id \in \mathsf{good}} \alpha_{id} = \sum_{id \in \mathsf{good}} (\alpha'_{id} - \eta) \geq (\alpha - 4\zeta' - \eta) \cdot \mu$$

□

We now lower bound the probability that the encoding algorithm succeeds under a random msg and random r. Recall that $\overline{\alpha}$ is the expected fraction of challenges $(id, \{Q_\ell\}_\ell)$ that $\mathcal{A}$ can respond to under a random choice of msg, r conditioned on the encoding algorithm not aborting before entering the offline challenge phase. Recall also that the encoding algorithm would abort if the number of predicted labels is fewer than $(\overline{\alpha} - \eta') \cdot \mu \cdot \sum_\ell s_\ell$.

Lemma 8.11 (Probability of successful encoding)**.** *Suppose $\overline{\alpha} > \eta'$. Under a random* msg *and a random r, the probability of successful encoding $\delta \geq (1 - \mathsf{negl}(\lambda) \cdot \frac{1}{T_\mathcal{A}} \cdot (\eta' - 4\zeta' - \eta)$.*

Proof. The encoding is successful iff the following hold:

1. The adversary indeed makes one or more queries of the form $H_\ell(\boldsymbol{\phi}^*, \cdot)$ where $\boldsymbol{\phi}^*$ is the digest of the database DB^* at the time of challege — this happens with $1 - \mathsf{negl}(\lambda)$ probability;
2. The encoding algorithm correctly guesses t^*. Conditioned on the above event, the probability of guessing correctly is at least $1/T_\mathcal{A}$ where $T_\mathcal{A}$ denotes the maximum number of random oracle queries made by $\mathcal{A}$.
3. The number of predicted labels is smaller than $(\overline{\alpha} - \eta') \cdot \mu \cdot \sum_\ell s_\ell$. Due to Claim 8.10, we have that conditioned on the encoding algorithm not aborting prior to entering the challenge phase,

$$\mathbb{E}[\sum_{id} P_{id}] \geq (\overline{\alpha} - 4\zeta' - \eta) \cdot \mu \cdot \sum_\ell s_\ell$$

where the randomness is taken over the choice of both msg and r.

Let p be the probability that this bad event happens conditioned on not aborting before entering the offline challenge phase. Henceforth, assume that $\eta' < \overline{\alpha}$. Due to Markov inequality, we have $(\overline{\alpha} - \eta') \cdot p + (1 - p) \geq \overline{\alpha} - 4\zeta' - \eta$, which implies that $1 - p \geq \eta' - 4\zeta' - \eta$.

Summarizing the above, the encoding is successful with probability at least

$$\delta \geq (1 - \mathsf{negl}(\lambda)) \cdot \frac{1}{T_{\mathcal{A}}} \cdot (1 - p) \geq (1 - \mathsf{negl}(\lambda)) \cdot \frac{1}{T_{\mathcal{A}}} \cdot (\eta' - 4\zeta' - \eta)$$

□

Theorem 8.12 (Space lower bound)**.** *Suppose $B = \omega(\log \lambda)$ and $\kappa = \omega(\log \lambda)$. Suppose we choose $\zeta' > 0$ to be an arbitrarily small constant. Let β be the probability that some deterministic adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ wins the $\mathsf{PoRepExpt}^{\mathcal{A}}(1^\lambda, S, \mu)$ game. Then, $|\mathsf{st}^{\mathcal{A}}| \geq (1 - o(1)) \cdot B \cdot (\beta - 5\zeta') \cdot \mu \cdot S$, where $o(1)$ is a sub-constant function in λ.*

Proof. Without loss of generality, we may assume that $\beta > 5\zeta'$ since otherwise, the theorem is trivially true. Observe also that $\beta \leq \overline{\alpha} + \mathsf{negl}(\lambda)$, since by definition, $\overline{\alpha}$ is equal to the probability that the adversary wins the $\mathsf{PoRepExpt}^{\mathcal{A}}(1^\lambda, S, \mu)$ game conditioned on having queried $H_\ell(\boldsymbol{\phi}^*, \cdot)$ for the digest $\boldsymbol{\phi}^*$ that corresponds to the challenge database. Moreover, the probability that $\mathcal{A}$ wins the game without having made such a query is negligibly small. Now, choose sufficiently small constants η and η' such that $\eta' \in (4\zeta' + \eta, 5\zeta')$, and $\overline{\alpha} > \eta'$, and run the encoding algorithm using these parameters to compress a randomly chosen msg using a random r.

We now analyze how much the message msg can be compressed if the encoding is successful. For each predicted label, instead of encoding the answer of the random oracle query which would have cost B bits, we only need to record the index of the identity, the index of the predicting query, and the starting offset of the predicted label within the relevent predicting query. The latter costs at most $\log_2 \mu + \log_2 T_{\mathcal{A}} + \log \log S + O(1)$ bits per predicted label. By Fact 4.1, we have

$$|\overline{\mathsf{msg}}| = |\mathsf{st}^{\mathcal{A}}| + |\mathsf{msg}| - (B - \log_2 \mu - \log_2 T_{\mathcal{A}} - \log \log S - O(1)) \cdot (\overline{\alpha} - \eta') \cdot \mu \cdot \sum_\ell s_\ell \geq |\mathsf{msg}| - \log \frac{1}{\delta}$$

where δ is the probability of successful encoding which we lower bounded in Lemma 8.11. Therefore, we have

$$\begin{aligned} |\mathsf{st}^{\mathcal{A}}| &\geq (B - \log_2 \mu - \log_2 T_{\mathcal{A}} - \log \log S - O(1)) \cdot (\overline{\alpha} - \eta') \cdot \mu \cdot \sum_\ell s_\ell - \log \frac{1}{\delta} \\ &\geq B \cdot (1 - o(1))(\overline{\alpha} - \eta') \cdot \mu \cdot \sum_\ell s_\ell - \log T_{\mathcal{A}} - \log \frac{1}{\eta' - 4\zeta' - \eta} - O(1) \end{aligned}$$

Since μ and $T_{\mathcal{A}}$ are polynomially bounded in λ, and $B = \omega(\log \lambda)$, we have that $B - \log_2 \mu - \log_2 T_{\mathcal{A}} - \log \log S - O(1) \geq (1 - o(1))B$. Based on our space allocation scheme, it must be that $\sum_\ell s_\ell \geq S$. Therefore, we conclude that

$$|\mathsf{st}^{\mathcal{A}}| \geq (1 - o(1)) \cdot B \cdot (\beta - \eta') \cdot \mu \cdot S \geq (1 - o(1)) \cdot B \cdot (\beta - 5\zeta') \cdot \mu \cdot S$$

□

9 Best Possible Recoverability

9.1 Definition of Extractor and Compression Algorithms

Extractor algorithm $\mathcal{E}^{\mathcal{A}_2}$. We define the following extractor algorithm.

$\underline{\mathcal{E}^{\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})}(1^\lambda, n, S, \mu, \boldsymbol{\phi}, \overline{\mathsf{DB}}^{\text{short}}; \rho)}$: // *all random coins consumed by $\mathcal{E}$ come from ρ*

- For each non-empty level ℓ (determined solely by n), initialize $P_\ell = \emptyset$, and $\overline{\mathsf{DB}}_\ell^{\text{ext}}$ to be an empty array where all positions are $\perp$.
- For each challenge identity id output by $\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})$, initialize $\widetilde{\mathbf{h}}_\ell$ to be empty, and repeat the following $\widetilde{T}^{\text{ext}}$ number of times to populate $\widetilde{\mathbf{h}}_\ell$, where $\widetilde{T}^{\text{ext}}$ is a super-polynomial function λ:
  - For each non-empty level ℓ: let $\widetilde{Q}_\ell$ be a freshly sampled multiset of κ indices from $G_\ell(id)$.
  - Rewind $\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})$ to the point when it has just submitted the j-th challenge identity id. Suppose $\mathcal{A}_2$ passes the offline phase, then feed it with the online challenges $\{\widetilde{Q}_\ell\}_\ell$.
  - Let $\{\phi_\ell^{\text{shard}}\}_\ell$ be the shard commitments submitted by $\mathcal{A}_2$ during the offline challenge phase, and let $\{(\mathsf{ans}_\ell, \widetilde{\boldsymbol{\pi}}_\ell^{\text{shard}})\}_\ell$ be $\mathcal{A}_2$'s answers for the online challenges $\{\widetilde{Q}_\ell\}_\ell$, where ans_ℓ is a vector of blocks corresponding to the challenged locations within the level ℓ, and $\widetilde{\boldsymbol{\pi}}_\ell^{\text{shard}}$ are the corresponding VC opening proofs. If $\mathsf{VC}.\mathbf{Vf}(\mathsf{crs}, 4s_\ell, \phi_\ell^{\text{shard}}, \mathsf{ans}_\ell, \widetilde{\boldsymbol{\pi}}_\ell^{\text{shard}}) = 1$ for all non-empty ℓ, then record the answers as follows. For each non-empty level ℓ: let $P_\ell \leftarrow P_\ell \cup \widetilde{Q}_\ell$, and populate $\widetilde{\mathbf{h}}_\ell[\widetilde{Q}_\ell] \leftarrow \mathsf{ans}_\ell$. Since the online challenges involve only the indices in $[3s_\ell+1 : 4s_\ell]$, only the part $\widetilde{\mathbf{h}}_\ell[3s_\ell + 1 : 4s_\ell]$ can be populated.

  At the end of the $\widetilde{T}^{\text{ext}}$ iterations, do the following. For every non-empty ℓ, for every position $i \in [3s_\ell + 1 : 4s_\ell]$ where $\widetilde{\mathbf{h}}_\ell[i]$ is correct (where the definition of correct is the same as in the encoding algorithm of Section 8.4), populate

$$\overline{\mathsf{DB}}_\ell^{\text{ext}}[G_\ell^{id}[i - 3s_\ell]] \leftarrow \widetilde{\mathbf{h}}_\ell[i] \oplus H_\ell^\rho(i, \widetilde{\mathbf{h}}_\ell[\mathsf{parents}(i)])$$

  where $\rho = (\boldsymbol{\phi}, id)$, and $G_\ell^{id} := G_\ell(id)$.
- Finally, parse $\overline{\mathsf{DB}}^{\text{short}} = \{\overline{\mathsf{DB}}_\ell^{\text{short}}\}_\ell$. For each non-empty level ℓ: call $\mathsf{DB}'_\ell \leftarrow \mathsf{EC}.\mathbf{Decode}(\overline{\mathsf{DB}}_\ell^{\text{ext}} \cup \overline{\mathsf{DB}}_\ell^{\text{short}})$, where $\cup$ is the following operation:

$$(\overline{\mathsf{DB}}_\ell^{\text{ext}} \cup \overline{\mathsf{DB}}_\ell^{\text{short}})[i] = \begin{cases} \overline{\mathsf{DB}}_\ell^{\text{short}}[i] & \text{if } \overline{\mathsf{DB}}_\ell^{\text{short}}[i] \neq \perp \\ \overline{\mathsf{DB}}_\ell^{\text{ext}}[i] & \text{o.w.} \end{cases}$$

  Now, from $\{\mathsf{DB}'_\ell\}_\ell$, reconstruct the database DB^{ext} and output the result.

Compression algorithm $\mathcal{C}^{\mathcal{A}}$. We now define the compression algorithm.

$\mathcal{C}^{\mathcal{A}}(1^\lambda, S, \mu, tr; \rho)$: *// all random coins consumed by $\mathcal{C}$ come from ρ*

- Replay the **Initialization** and **Queries** with $\mathcal{A}_1$ using the random coins contained in tr, and obtain n, DB^*, $\boldsymbol{\phi}$, and $\mathsf{st}^{\mathcal{A}}$;
- Execute $\mathcal{E}^{\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})}(1^\lambda, n, S, \mu, \boldsymbol{\phi}, \_; \rho)$ except the final $\mathsf{EC}.\mathbf{Decode}$ step, and obtain $\{\overline{\mathsf{DB}}_\ell^{\text{ext}}\}_\ell$.
- For each non-empty level ℓ, suppose that $\overline{\mathsf{DB}}_\ell^{\text{ext}}$ has k_ℓ locations populated. Then, choose $\max(0, 2^\ell - k_\ell)$ locations that are $\perp$ in $\overline{\mathsf{DB}}_\ell^{\text{ext}}$, and populate those positions in $\overline{\mathsf{DB}}_\ell^{\text{short}}$ using the correct values from $\overline{\mathsf{DB}}_\ell^*$, which can in turn be computed from $\overline{\mathsf{DB}}^*$. All other locations in $\overline{\mathsf{DB}}_\ell^{\text{short}}$ are set to $\perp$.
- Output $\overline{\mathsf{DB}}^{\text{short}} := \{\overline{\mathsf{DB}}_\ell^{\text{short}}\}_\ell$. We will later show that $n - (1-\epsilon) \cdot S \cdot \mu$ bits are sufficient for encoding $\overline{\mathsf{DB}}^{\text{short}}$.

Fiat-Shamir extractor $\mathcal{E}_{\text{fs}}^{\mathcal{A}}$. We define another extractor that extracts a replication encoding by reprogramming the random oracle FS used for determining the offline challenges. This extractor $\mathcal{E}_{\text{fs}}^{\mathcal{A}}$ will serve as an aid in our proofs later.

$\mathcal{E}_{\text{fs}}^{\mathcal{A}}(1^\lambda, S, \mu, tr)$ is a randomized algorithm that works just like the extractor in our encoding algorithm of Section 8.4 except with the following changes:

- The encoding algorithm of Section 8.4 needs to guess t^*, but here we no longer need to, and thus we will never abort prior to entering the challenge phase.
- The encoding algorithm of Section 8.4 uses msg to determine the random oracles $\{H_\ell(\boldsymbol{\phi}^*, \cdot)\}_\ell$ with msg. Here, we no longer have msg, and the random oracle queries are answered as follows. Before the extractor starts rewinding $\mathcal{A}$, all random oracle queries will be answered using the coins contained in tr. After the extractor starts rewinding $\mathcal{A}$, any fresh random oracle query will be answered using the extractor's own internal randomness — here fresh means any random oracle query $\mathcal{A}$ has not made upto the rewinded point, including the query $\mathsf{FS}_\ell(\phi_\ell^{\text{shard}}, id)$ itself for the challenge pair $(\phi_\ell^{\text{shard}}, id)$ itself.

Finally, if no collisions are detected, the extractor $\mathcal{E}_{\text{fs}}$ will output an array of labels $\mathbf{h}_\ell^{id}$ for each non-empty level ℓ and each id that is a challenge identity under a tr in which $\mathcal{A}$ successfully answers the audits for all μ challenge identities.

9.2 Analysis

We will consider the following experiment that is a slight modification of the experiment in Definition 1, where we additionally run the extractor $\mathcal{E}_{\text{fs}}$ to aid the proof.

- $b, \phi, \mathsf{DB}^*, \mathsf{st}^{\mathcal{A}}, tr \leftarrow \mathsf{RecvExpt}^{\mathcal{A}}(1^\lambda, S, \mu)$;
- $\{\mathbf{h}_\ell^{id}\}_{\ell, id} \leftarrow \mathcal{E}_{\text{fs}}^{\mathcal{A}}(1^\lambda, S, \mu, tr)$;
- $\rho \xleftarrow{\$} \{0,1\}^{|\rho|}$;
- $\mathsf{DB}^{\text{short}} \leftarrow \mathcal{C}^{\mathcal{A}}(1^\lambda, S, \mu, tr; \rho)$;
- $\mathsf{DB}^{\text{ext}} \leftarrow \mathcal{E}^{\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})}(1^\lambda, n, S, \mu, \epsilon, \phi, tr, \mathsf{DB}^{\text{short}}; \rho)$.

Throughout the proof, we assume $\eta \in (0,1)$ is an arbitrarily small constant.

Claim 9.1. *Lemma 8.5 still holds in the above experiment for the extractor $\mathcal{E}_{\text{fs}}$. In other words, except with negligible probability, if $\mathcal{A}$ passes the audit, then for every challenge identity id, every non-empty level ℓ, in the reconstructed label array $\mathbf{h}_\ell^{id}$ output by $\mathcal{E}_{\text{fs}}^{\mathcal{A}}$, at least $1-\eta$ fraction of the positions must be correct.*

Proof. The proof is the same as the proof of Lemma 8.5, except that for Claim 8.6, we only need to rely on the collision resistance of VC against quasi-polynomial-time adversaries, and we no longer need to consider H_{vc} as a random oracle. This is because in the encoding algorithm of Section 8.4, we cared only about bounding the number of random oracle calls made by the encoding algorithm, but not its running time, since the encoding algorithm takes an exponentially-long random r and msg as input. Here, however, the entire experiment is quasi-polynomially bounded. □

Claim 9.2. *Except with negligible probability, if the adversary passes the audit for all μ challenge identities, then for each level ℓ, every challenge identity id will correctly populate at least $(1-2\eta)\cdot s_\ell$ positions in $\overline{\mathsf{DB}}_\ell^{\text{ext}}$.*

Proof. Using almost the same (but a slightly simpler) argument as the proof of Claim 9.1, we know that except with negligible probability, if $\mathcal{A}_2(\mathsf{st}^{\mathcal{A}})$ passes the audit, then for each challenge id, each ℓ, at least $(1-\eta)\cdot s_\ell$ extracted positions in each $\widetilde{\mathbf{h}}_\ell^{id}$ agree with the shard commitment ϕ_ℓ^{id} submitted by $\mathcal{A}_2$ in the offline challenge phase, where we use the superscript id to mean the corresponding variables pertaining to challenge identity id. The proof is actually slightly simpler than that of Claim 9.1 because here the challenges are sampled interactively, and we do not need to handle the $T^{\mathcal{A}}$ loss due to the Fiat-Shamir paradigm.

Henceforth, we ignore the negligible probability that for some position i, both $\widetilde{\mathbf{h}}_\ell^{id}[i]$ and $\mathbf{h}_\ell^{id}[i]$ are populated, but $\widetilde{\mathbf{h}}_\ell^{id}[i] \neq \mathbf{h}_\ell^{id}[i]$ — since if this event happens, then we can construct a quasipolynomial time algorithm that can find hash collisions. Due to Claim 9.1, for each id and ℓ, among the $1-\eta$ fraction of positions that agree with ϕ_ℓ^{id}, at most η fraction is incorrect. Therefore, for each level ℓ, every challenge identity can correctly populate at least $(1-2\eta)\cdot s_\ell$ positions in $\overline{\mathsf{DB}}_\ell^{\text{ext}}$. □

Fix some level ℓ. Claim 9.2 shows that the extractor $\mathcal{E}^{\mathcal{A}_2}$ can populate many positions in $\overline{\mathsf{DB}}_\ell^{\text{ext}}$ from each challenge identity. However, the positions populated by the different identities may have overlap. Next, we want to show that the overlap cannot be very large. Specifically, we prove that if the adversary is polynomially bounded, it cannot find μ number of challenge identities whose respective sampled subsets $G_\ell(id)$ overlap significantly.

Lemma 9.3 (Small overlap among multiple identities)**.** *Fix some level ℓ, and let $\eta \in (0,1)$ be a sufficiently small constant. Suppose $s_\ell > \omega(\log\lambda)$, and the redundancy of* EC *is at least $R \geq (1+\eta)e^{2/\eta}$. Then, except with* $\mathsf{negl}(\lambda)$ *probability, no adversary that makes at most $T^{\mathcal{A}} = \mathsf{poly}(\lambda)$ number of random oracle queries can output μ identities $id_1,\ldots,id_\mu$, such that $G_\ell(id_1)\cup G_\ell(id_2)\cup \ldots \cup G_\ell(id_\mu) < \min\left(2^\ell, (1-\eta)\cdot s_\ell\cdot\mu\right)$.*

Proof. Henceforth, if the adversary outputs some challenge identity id but it has not queried $G_\ell(id)$, we simply assume that the adversary is given a query to id for free. In this way, the total number of random oracle queries is at most $T := T^{\mathcal{A}} + \mu$.

Henceforth, let $n_\ell = 2^\ell$, let $n' = s_\ell\cdot\mu$, and let $\theta = \frac{R\cdot n_\ell}{\min(n_\ell,(1-\eta)n')}$ where R is the redundancy parameter of the erasure code. Given a fixed subset $I \subseteq [R\cdot n_\ell]$ of size at most $\min(n_\ell,(1-\eta)n')$, we say that some identity *id falls within I* iff $G_\ell(id)\subseteq I$. We have

$$\Pr\left[\text{at least } \mu \text{ queried identities fall within } I\right] \leq \left(\frac{1}{\theta}\right)^{n'} \cdot \binom{T}{\mu}$$

We have that

$$\begin{aligned}
&\Pr\left[\exists I: \text{ at least } \mu \text{ queried identities fall within } I\right]\\
\leq& \left(\frac{1}{\theta}\right)^{n'} \cdot \binom{T}{\mu} \cdot \binom{R\cdot n_\ell}{\min(n_\ell,(1-\eta)n')}\\
\leq& \left(\frac{1}{\theta}\right)^{n'} \cdot \left(\frac{e\cdot T}{\mu}\right)^{\mu} \cdot (e\cdot\theta)^{(1-\eta)n'}\\
\leq& \Bigg(\underbrace{\left(\frac{1}{\theta}\right)^{0.5\eta\cdot s_\ell}\cdot\left(\frac{eT}{\mu}\right)}_{(\diamondsuit)}\Bigg)^{\mu} \cdot \Bigg(\underbrace{\left(\frac{1}{\theta}\right)^{1-0.5\eta}\cdot(e\theta)^{1-\eta}}_{(\clubsuit)}\Bigg)^{s_\ell\cdot\mu}
\end{aligned}$$

Given that $R \geq (1+\eta)e^{2/\eta}$ and $T = \mathsf{poly}(\lambda)$, as long as $s_\ell = \omega(\log \lambda)$, we have that ($\diamond$) is negligibly small in λ. One can also mechanically verify that if $\theta \geq e^{2/\eta}$, then the ($\clubsuit$) part is at most 1. Further, if $R \geq (1+\eta) \cdot e^{2/\eta}$, then $\theta \geq e^{2/\eta}$ must be guaranteed. □

Theorem 9.4 (ϵ-best-recoverability)**.** *Let $\epsilon \in (0,1)$ be an arbitrarily small constant. Suppose $S = \omega(\log^2 \lambda)$, $\kappa = \omega(\log \lambda)$, and $R \geq (1+\epsilon)e^{O(1/\epsilon)}$. Then, our construction in Section 6.1 satisfies ϵ-best-recoverability.*

Proof. Since $S = \omega(\log^2 \lambda)$, it is possible to choose the super-constant function in $s_{\text{lb}} = \gamma \cdot 2^L/\omega(1) \log n$ to be sufficiently small such that $s_\ell = \omega(\log \lambda)$ for all ℓ — this condition will be needed for invoking Lemma 9.3 below.

Recall that Claim 9.2 says that with each identity the extractor $\mathcal{E}^{\mathcal{A}_2}$ must be able to populate many positions in $\overline{\mathsf{DB}}_\ell^{\text{ext}}$. Further, by construction, for each identity id, these populated locations must be among $G_\ell(id)$. Imagine that we iterate through each challenge id one by one. For each id, let Δ_{id} be the number of locations in $\overline{\mathsf{DB}}_\ell^{\text{ext}}$ that can be populated by id but overlapping with those already populated by earlier identities. Therefore, the total number of distinct populated locations at the end is $(1-2\eta)s_\ell \cdot \mu - \sum_{id} \Delta_{id}$. By Lemma 9.3, with all but negligible probability, $\sum_{id} \Delta_{id} \leq \eta \cdot s_\ell \cdot \mu$. Therefore, combining Claim 9.2 and Lemma 9.3, we have that except with negligible probability, given that all μ challenge identities pass the audit, the extractor must be able to populate at least $(1-3\eta)s_\ell \cdot \mu$ number of *distinct* locations in $\overline{\mathsf{DB}}_\ell^{\text{ext}}$.

Additionally, due to Claim 9.1, the number of locations in $\overline{\mathsf{DB}}_\ell^{\text{ext}}$ extracted by both $\mathcal{E}^{\mathcal{A}_2}$ and $\mathcal{E}_{\text{fs}}^{\mathcal{A}}$ must be at least $(1-4\eta)s_\ell \cdot \mu$. Since we assume that VC is collision resistant against quasi-polynomial-time adversaries, except with negligible probability, if a location in $\overline{\mathsf{DB}}_\ell^{\text{ext}}$ is extracted by both $\mathcal{E}^{\mathcal{A}_2}$ and $\mathcal{E}_{\text{fs}}^{\mathcal{A}}$, the extracted values must agree, and due to Claim 9.1, the extracted value must also be correct.

Now, recall that the compression algorithm $\mathcal{C}^{\mathcal{A}}$ will simply populate sufficiently many additional locations in $\overline{\mathsf{DB}}_\ell^{\text{ext}}$ to complement what $\mathcal{E}^{\mathcal{A}_2}$ can populate, such that in total, n_ℓ locations will be populated for level ℓ. Based on the property of the erasure coding scheme EC, from these n_ℓ locations in the encoded level, we can correctly decrypt the orginal blocks that belong to the level.

We now analyze the size of the summary $\overline{\mathsf{DB}}^{\text{short}}$ output by the compressor $\mathcal{C}^{\mathcal{A}}$. Except with negligible probability, the number of locations $\mathcal{C}^{\mathcal{A}}$ needs to populate in a fixed level ℓ is at most $n_\ell - (1-5\eta)s_\ell \cdot \mu$. Due to our choice of $\{s_\ell\}_\ell$,

$$
\begin{aligned}
|\overline{\mathsf{DB}}^{\text{short}}| &\leq \sum_{\ell \text{ non-empty}} (n_\ell - (1-5\eta)s_\ell \cdot \mu) \\
&\leq \left(\sum_\ell n_\ell\right) - (1-5\eta)\mu \cdot \left(\sum_\ell s_\ell\right) \\
&\leq n - (1-5\eta)\mu \cdot \sum_{\ell \text{ non-empty, large}} \sum_\ell s_\ell \\
&\leq n - (1-5\eta)\mu \cdot (1-o(1)) \cdot S && \text{(by Fact 6.1)} \\
&\leq n - (1-6\eta)\mu \cdot S
\end{aligned}
$$

To satisfy ϵ-best recoverability, we can simply choose $\eta = \epsilon/6$. □

Acknowledgments

This work is in part supported by NSF awards 2212746, 2044679, a Packard Fellowship, and a generous gift from the late Nikolai Mushegian. Changrui Mu is supported by the National Research Foundation, Singapore, under its NRF Fellowship program, award number NRF-NRFF14-2022-0010. We gratefully acknowledge Ashrujit Ghoshal for helpful discussion on techniques for proving time-space tradeoff. We also acknowledge Hao Chung for helpful discussons on data availability protocols.

References

[ABP18] Joel Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In *Eurocrypt*, 2018.

[ABSBK21] 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*, page 279–298, Berlin, Heidelberg, 2021. Springer-Verlag.

[ACDW20] Akshima, David Cash, Andrew Drucker, and Hoeteck Wee. Time-space tradeoffs and short collisions in merkle-damgard hash functions. In *CRYPTO*, 2020.

[AGL22] 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.

[AKK09] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In *ASIACRYPT*, 2009.

[AKLY24] 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.

[BBC+24] 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.

[BBHR18] 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.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *ACM symposium on Theory of computing (STOC)*, 2013.

[BCG24] Annalisa Barbara, Alessandro Chiesa, and Ziyi Guan. Relativized succinct arguments in the ROM do not exist. *IACR Cryptol. ePrint Arch.*, page 728, 2024.

[BJO09] Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: theory and implementation. In *CCSW*, 2009.

[BNNP25] Dan Boneh, Joachim Neu, Valeria Nikolaenko, and Aditi Partap. Data availability sampling with repair. Cryptology ePrint Archive, Paper 2025/1414, 2025.

[BS80] Jon Louis Bentley and James B Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

[CBK+24] 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.

[CDG+24] 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*, page 158–188, Berlin, Heidelberg, 2024. Springer-Verlag.

[CDGS18] Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John P. Steinberger. Random oracles and non-uniformity. In *EUROCRYPT (1)*, pages 227–258. Springer, 2018.

[CGW25] Alessandro Chiesa, Ziyi Guan, and Yuetian Wu. Unpublished manuscript. Private conversation with Ziyi Guan, 2025.

[CKO14] 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.

[CSK25] Seunghyun Cho, Eunyoung Seo, and Young-Sik Kim. Locally recoverable data availability sampling. Cryptology ePrint Archive, Paper 2025/1851, 2025.

[CT05] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *Proceedings of the 19th International Conference on Distributed Computing*, DISC'05, page 503–504, Berlin, Heidelberg, 2005. Springer-Verlag.

[DGK17] 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.

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-np and applications. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1057–1068, 2022.

[DTT10] 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.

[DVW09] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *Theoretical Cryptography Conference (TCC)*, 2009.

[EGS75] Paul Erdoes, Ronald L. Graham, and Endre Szemeredi. On sparse graphs with dense long paths. Technical report, 1975.

[Fis18] Ben Fisch. Poreps: Proofs of space on useful data. *IACR Cryptol. ePrint Arch.*, page 678, 2018.

[Fis19] 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*, page 324–348, Berlin, Heidelberg, 2019. Springer-Verlag.

[FLLY24] 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.

[GGKL21] 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, page 2404–2423, USA, 2021. Society for Industrial and Applied Mathematics.

[GXQZ25] Yanpei Guo, Alex Luoyuan Xiong, Wenjie Qu, and Jiaheng Zhang. Data availability for thousands of nodes. Cryptology ePrint Archive, Paper 2025/865, 2025.

[HGR07] 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, page 139–146, New York, NY, USA, 2007. Association for Computing Machinery.

[HSW24a] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Foundations of data availability sampling. *IACR Commun. Cryptol.*, 1(4):34, 2024.

[HSW24b] 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.

[JK07] Ari Juels and Burton Kaliski. Pors: proofs of retrievability for large files. In *ACM CCS*, 2007.

[Kup10] A. Kupcu. Efficient cryptography for the next generation secure cloud. PhD thesis, Brown University, 2010.

[KZG10] 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.

[Mer89] Ralph C. Merkle. A certified digital signature. In G. Brassard, editor, *Proc. CRYPTO '89*, volume 435 of *LNCS*, pages 218–238. Springer-Verlag, 1989.

[NNT23] 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, page 180–197, New York, NY, USA, 2023. Association for Computing Machinery.

[Pie19] 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.

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1045–1056, 2022.

[SSP13] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[SW08] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *Asiacrypt*, 2008.

[SW13] Hovav Shacham and Brent Waters. Compact proofs of retrievability. *J. Cryptol.*, 26(3):442–483, July 2013.

[Unr07] Dominique Unruh. Random oracles and auxiliary input. In *Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO'07, page 205–223, Berlin, Heidelberg, 2007. Springer-Verlag.

[WPSP24] Weijie Wang, Charalampos Papamanthou, Shravan Srinivasan, and Dimitrios Papadopoulos. Dynamic zk-SNARKs. Cryptology ePrint Archive, Paper 2024/1566, 2024.

[WTS+18] 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.

[YPA+22] 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.

A Achieving Polynomial-Time Extraction under Relaxed Definitions

Relaxed extraction notion: weak recoverability. We now relax our best possible recoverability notion using essentially the same relaxation adopted in prior work [HSW24a], which allows us to get a strict polynomial-time extractor like in prior work [HSW24a].

Specifically, we modify the earlier RecvExpt experiment as follows, parametrized by some parameter τ which denotes the number of invocations per challenge identity.

- **Initialization, Queries.** Same as before.
- **Challenge.** The adversary $\mathcal{A}$ specifies μ identities denoted $id_1, \ldots, id_\mu$, such that $\mu \cdot S \geq (1+\epsilon)n$. Now, for each $j \in [\mu]$, the challenge invokes τ independent instances of the **Audit** protocol using $(\mathsf{crs}, \phi, id_j)$ as input. Let Tr denote the collection of all transcripts of all $\mu \cdot \tau$ instances of **Audit**.

Definition 4 (ϵ-weak-recoverability)**.** We say that an iDDA scheme satisfies ϵ-weak-recoverability, iff there exists some τ that is polynomially bounded in λ, and a polynomial-time extractor $\mathcal{E}$ such that except with negligible probability, the following holds: if the adversary $\mathcal{A}$ succeeds in all invocations of **Audit** in the above experiment, then $\mathcal{E}(\mathsf{Tr}) = \mathsf{DB}$, where DB is the true cumulative database determined by the initial database and all the updates submitted by the adversary $\mathcal{A}$.

The above notion requires the adversary to succeed in all invocations of **Audit** for the extraction to be successful. Like in Hall-Andersen et al. [HSW24a], we can relax this requirement on the adversary by assuming that the adversary succeeds τ out of T invocations (called subset soundness in their paper). Using the same argument as their Lemma 1 [HSW24a], we can show that our ϵ-weak-recoverability for an always successful adversary implies ϵ-weak-recoverability for an τ-out-of-T-successful adversary, with an $\binom{T}{\tau}$ factor loss in the failure probability.

Proof of ϵ-weak-recoverability. Our proofs in Section 9 can be easily modified to show ϵ-weak-recoverability with a strict polynomial-time extractor. In particular, since ϵ-weak-recoverability cares only about extracting the entire database (and not partial information) when given sufficiently many nodes that can succeed in the **Audit** protocol, we no longer need to worry about the compression algorithm in the proof. Further, the original extractor $\mathcal{E}$ is now essentially baked into the definition, which works by invoking the μ adversarial nodes each $\tau \geq C \cdot S/\kappa \geq C \cdot s_\ell/\kappa$ number of times where κ is the number of indices sampled for the challenge per level, and C is a suitably large constant. However, with the new definition, the extractor is only required to be successful if $\mathcal{A}$ succeeds in answering all **Audit** instances. Under this very strong requirement, it is now much easier to show that Claim 9.2 still holds. Essentially, assuming that VC is secure and no hash collisioins occur, this boils down to showing that if we sample $\kappa \cdot \tau$ random challenges from $[s_\ell]$, except with negligible probability, $(1-\eta)s_\ell$ fraction of the indices will be covered, where η is a suitably small constant. Specifically, this probability is upper bounded by $(1-\eta)^{C\cdot\kappa\cdot\tau} \cdot \binom{s_\ell}{(1-\eta)s_\ell}$. It is not hard to see that if $e \cdot (1-\eta)^C < 1$, then this expression is upper bounded by $\exp(-\Omega(s_\ell))$.