The following publication Duan, H., Du, Y., Zheng, L., Wang, C., Au, M. H., & Wang, Q. (2023). Towards practical auditing of dynamic data in decentralized storage. IEEE Transactions on Dependable and Secure Computing, 20(1), 708-723 is available at https://doi.org/10.1109/TDSC.2022.3142611.

Towards Practical Auditing of Dynamic Data in Decentralized Storage

Huayi Duan, Yuefeng Du, Leqian Zheng, Cong Wang, Man Ho Au, Qian Wang

Abstract—Decentralized storage (DS) projects such as Filecoin are gaining traction. Their openness mandates effective auditing mechanisms to assure users that their data remains intact. A blockchain is typically employed here as an unbiased public auditor. While the case for static data is relatively easy to handle, on-chain auditing of dynamic data with practical performance guarantees is still an open problem. Dynamic Proof-of-Storage (PoS) schemes developed for conventional cloud storage are not applicable to DS, since they require large storage proofs and/or large auditor states that are unmanageable by resource-constrained blockchains. To fill the gap, we propose a family of dynamic on-chain auditing protocols that can produce concretely small auditor states while retaining the compact proofs promised by static PoS schemes. Our design revolves around a set of succinct data structures and optimization techniques for index information management. With proper instantiation and realistic parameters, our protocols can achieve 0.25MB on-chain state and 1.2KB storage proof for the auditing of 1TB data, outperforming previous dynamic PoS schemes that are adaptable for DS by orders of magnitude. As another practical contribution, we introduce a data abstraction layer that allows one to deploy the auditing protocols on arbitrary storage systems hosting dynamic data.

Index Terms—Decentralized Storage, Blockchain, Data Integrity Auditing, Data Dynamism

1 Introduction

Decentralized storage (DS) emerges as an appealing alternative to commercial cloud storage. It consists of a peer-to-peer network that anyone can join to store and serve data for others. The high-profile projects such as Filecoin, Sia and Storj envision exciting storage markets that connect people with the need to outsource data and those seeking to monetize unused disk space.

The openness of DS mandates convincing technical means to assure users of the integrity of their remotely stored data. Almost all DS projects today have built-in data auditing mechanisms (see detailed discussion in Section 2.1). They employ some auditors to routinely verify random portions (chunks) of the outsourced data at storage nodes, which will get remunerated or fined according to the audit results. Implementing storage auditing is imperative to the ecosystem's health and correct functioning, as the lack of such an assurance will keep wary users away.

Delegating the mission-critical auditing function to any single party violates the principle of decentralization. Most DS projects [2], [18], [27], [35] rely on a general-purpose blockchain to serve as an unbiased public auditor. It verifies storage proofs submitted by the nodes, and resolve payment and potential disputes between parties in a fair manner. This architecture, which we call *on-chain auditing*, requires storage proofs and any auditor states to be as small as possible, because a blockchain is not well-suited for handling large data and computation. Classic Proof of Storage (PoS) schemes [5], [39] offer a promising solution to meet such requirements as they can produce compact proofs, but their efficiency is only guaranteed for static data.

The challenge of data dynamism. To be as usable as cloud storage, a DS system should allow users to update their data at will. For example, many users back up their working folders to 3rd-party storage services like Dropbox where files are updated on a regular basis. However, the unique properties and requirements of on-chain auditing make the handling of dynamic data a challenging task.

In the DS community, the most prominent project Filecoin builds its storage proof system around zkSnark [10], which produces extremely short proofs and is very appealing to on-chain auditing. But this approach is not amenable to dynamic data, as a single update may trigger an expensive setup procedure (see more in Section 2.1). Designing efficient updatable proof system is indeed scheduled as an open problem by the team [26].

On the academic side, a list of dynamic PoS schemes have been proposed for traditional cloud storage. Some of them are inherently inapplicable to the architecture of on-chain auditing for security reasons. The designs based on precomputed verification tokens (e.g., [6], [47]) require the auditor to store secret materials, which is at odds with the transparency of public blockchain. Another line of schemes (e.g., [41], [42]) place original data chunks inside storage proofs, creating the possibility for storage nodes to pass audits without actually storing data.

Dynamic PoS schemes that are potentially adaptable to the context of DS can be divided into two main classes.

The first class of designs [21], [48] combines homomorphic tags and authenticate data structure (ADS) to safeguard the outsourced data while allowing efficient update with sublinear costs. Their major drawback is the large proof size. The use of ADS results in $O(c \log n)$ storage proofs, where n is the total number of chunks and c is the amount of them challenged in each audit; the concrete size becomes 100s of KBs for a typical security level. To put the number into perspective, note that the block sizes of Ethereum are almost always under 60KB [25]; in other words, a storage proof here cannot fit into even an entire block.

The second class of designs (e.g., [30], [45], [53]) can

produce shorter proofs comparable to that of classic static PoS schemes [5], [39]. But they require the auditor to maintain $O(n)$ dynamic states with high update costs, which are expensive and unreasonable for on-chain auditing. This precludes their use in real-world DS networks with growing user base and data volumes.

To sum up, how to practically audit dynamic data in the emerging context of DS is still a largely unexplored problem.

Our contribution. We examine the design space of dynamic PoS and observe that the technicalities revolve around the handling of index information. With a proper index translation mechanism, it is possible to transform a static PoS into a dynamic one with the former's compact storage proofs retained. This mechanism introduces a non-trivial *index state* to parties in an auditing protocol, with the key technical challenge being how to manage the index state such that it can be practically maintained by a resource-sensitive public blockchain where *every byte counts*. Our contributions are summarized as follows.

- We design two generic on-chain auditing protocols for dynamic data. The first protocol OAD achieves an on-chain index state of $4n$ bytes, the smallest ever concrete size for any auditing schemes with $O(n)$ auditor state. The second protocol OAD^+ further reduces on-chain state to $10n/m$ bytes, where m is a tunable parameter. When instantiated with an efficient PoS construction, both protocols guarantee short, constant-size proofs.
- We motivate and design a data abstraction layer. It decouples the auditing functionality from data storage, allowing auditing protocols to be deployed on arbitrary underlying storage systems that have varying data formats and organization with minimal overheads.
- We confirm the practicality of our designs with a detailed evaluation. For 1TB file and 128-bit security, the proof size is 0.13KB for OAD and 1.2KB for OAD^+; in comparison, ADS-based designs generate proofs over 100KB. In terms of on-chain state, OAD^+ can achieve a size as small as 0.25MB, while the best alternative solution would require 4GB state for a comparable proof size.

2 Related Work

2.1 Decentralized Storage

We start by introducing major DS projects and discussing insufficiencies of their storage auditing mechanisms.

Storj[1] introduces a role called satellite that is entrusted with a collection of critical services including data auditing [8]. A satellite regularly requests random portions of outsourced files from storage nodes and audits the data with error correction algorithms. This centralized approach places undue trust on satellites, violating DS's fundamental principle of decentralization.

Sia [18] takes the more desired decentralized approach of on-chain auditing. It authenticates users' data with Merkle Hash Tree (MHT) [36] and stores tree roots on the blockchain. To claim rewards, storage nodes must submit randomly sampled data chunks (tree leaves) with respective verification paths to the blockchain. Such a simple MHT-based scheme faces scalability issues, as large data chunks and proofs need to be transmitted over the network during the consensus process and eventually recorded on the blockchain. It also has an inherent security flaw: once a chunk is audited and stored as on-chain data, storage nodes can discard it locally and retrieves it from the blockchain for future audits whenever necessary, violating the expected local storage guarantee. This reveals that for on-chain auditing the storage proofs must be *chunk-free*, i.e., without carrying the original data chunks.

Filecoin [2] is the most prominent DS project to date. It envisions two open marketplaces for data storage and retrieval. As its salient feature, Filecoin leverages storage proofs themselves to design an energy-efficient consensus protocol. In particular, it uses zkSnark [11] to create succinct and easy-to-verify proofs. Here, files are organized as sectors, and each audit requests two zkSnark proofs: one to show that the sector in question is correctly encoded as a unique replica and a MHT over the replica is constructed; the other to prove that a challenged chunk of the sector is verified with its Merkle proof. This design is ideal for on-chain auditing owning to its unbeatable verification efficiency. However, zkSnark is also known for its excessive prover-side and preprocessing costs; numerous efforts are still made by Filecoin team to optimize these aspects in their proof system[2]. Such deficiencies make it onerous to handle dynamic data, because a single data update can lead to a fresh setup of the sector and zkSnark circuits, the overheads of which are prohibitive for many practical use cases. Indeed, supporting efficient data updates is still listed as an open problem by the Filecoin community[3].

Koppercoin [35] and Audita [27] are both blockchain consensus protocols based on static PoS scheme [39] and [5], respectively. Users cannot change data after the initial setup. Audita differs from all aforementioned projects in that it is not explicitly designed as a storage marketplace, and its storage nodes are solely incentivized by mining awards.

Recently, Campanelli et al. [14] propose a verifiable decentralized storage scheme supporting dynamic data. However, relying on heavy cryptography (subvector commitment), the scheme is currently more of theoretical than practical interest.

2.2 Dynamic Proof of Storage

We now review representative designs on dynamic PoS, explaining why they are not applicable to the context of DS.

Partially dynamic schemes. Ateniese et al. [6] propose a partially dynamic provable data possession (PDP) scheme that supports limited modification, deletion and appending of file chunks. It works by pre-computing verification tokens for a fixed number of future challenges. Each storage audit will consume one token. Applying a similar idea, a later design [47] additionally enables data error localization and addresses a privacy issue in the setting of third-party auditor. A general drawback of this approach is that each update requires recomputing all remaining, unused tokens.

1. The Storj project: https://www.storj.io

2. Filecoin Proving Subsystem: https://github.com/filecoin-project/rust-fil-proofs

3. Filecoin Research: https://github.com/filecoin-project/research

More importantly, these schemes require the auditor to keep verification states that are supposed to be hidden from the storage server, i.e., secret token-deriving keys or verification tokens themselves. Their security properties will be immediately broken when adapted to on-chain auditing, as storage nodes can access all verification states on the blockchain.

ADS-based schemes. In their seminal work [22], Erway et al. introduce a public PoS scheme called DPDP that supports full data dynamism. It combines homomorphic tags and a rank-based authenticated skip list to protect data integrity. Most performance metrics, including computation and communication cost of audit and update, inherit the logarithmic complexity of the data structure. The scheme by Wang et al. [48] combines aggregate short signature with MHT, achieving better concrete efficiency than DPDP with the same asymptotic complexity.

These ADS-based designs may work perfectly in traditional cloud storage, but they do not fit on-chain auditing for decentralized storage. As mentioned, their $O(c \log n)$ proofs can be practically too large (e.g., hundreds of KBs for a reasonable security level) to be transmitted, verified, and stored by a blockchain network. A follow-up design of DPDP [23] optimizes the data structure but the improvement is not significant (i.e., less than 50% reduction in proof size). Using a depth-fixed RSA tree [37] could avoid the $\log n$ factor, but only to boost the constant factor (multiple RSA group elements) and result in even larger proofs. Recently, Guo et al. propose a authenticated 2-3 tree for efficient auditing of multiple replica of the same file [29]. They compress a storage proof by pruning redundant nodes from the verification paths of multiple challenged chunks. Yet, the savings can not be substantial given the random nature of challenge generation, especially when there is a large amount of chunks. Essentially, the fact that verification paths of tree-like ADS are not *aggregatable* will inevitably lead to much inflated storage proofs. A recent survey revisits this generic approach in greater details [24].

Another line of works targets the stronger notion of Proof of Retrievability (POR). The main challenge here is to prevent the storage server from discarding blocks whose erasure codes are revealed during updates. Cash et al. [15] propose a theoretical construction from the black-box application of oblivious RAM [28]. Based on the idea of delaying updates and batch processing, the Iris system [42] and the schemes by Shi et al. [41] provide more practical solutions. However, their storage proofs are not chunk-free (see Section 2.1) and they experience the inefficiency of ADS-based design because of their reliance on MHT.

Recently, Anthoine et al. propose a family of dynamic POR schemes based on linear algebraic verification [4]. The design can achieve minimal storage overhead on the server side, but at the expense of dramatically increased server computation, communication and verification costs. In particular, the storage proofs consist of $O(\sqrt{n})$-sized matrices and the verification involve matrix multiplication. The construction for public verifiability is even worse for the required group operations. Apparently, this line of schemes are ill-suited for on-chain auditing.

Schemes with large auditor state. A series of dynamic schemes achieve shorter storage proofs at the cost of large $O(n)$ auditor states. The scheme by Hao et al. [30] requires the auditor to keep all authentication tags locally, leading to a very large constant factor (hundreds of bytes, the size of RSA group elements) for the state.

Similar to ours, several schemes [32], [51], [53] apply the idea of chunk index translation to ensure that an update will not cause an excessive number of tags to be recalculated. They record the mappings of two types of index information sequentially in a simple table structure, which is the state the auditor should keep. While achieving smaller auditor state size, these designs impose $O(n)$ update cost on the state. This may be justifiable for off-chain parties but too expensive for an on-chain auditor. A more recent scheme [50] employs a tree-based index management design; while achieving $O(\log n)$ overall update efficiency, its data structure is still suboptimal regarding space efficiency.

Another two related schemes [40], [45] use version and timestamps instead of index information to generate authentication tag. The auditor then maintains a set of ordinary linked lists that records data version and timestamps as verification state. One flaw here is that unlike index, data version and timestamp are not necessarily unique among chunks, e.g., initially all chunks have the same version and timestamp. The lack of uniqueness guarantee renders these designs insecure. Moreover, the auditor state size can have a constant factor of tens of bytes due to the data structure overhead in practical implementation.

Regarding the idea of using succinct linked list for index translation, the design by Barsoum and Hasan [9] is the closest one to ours. It has $8n$-byte auditor state and allows constant-time update. However, in their scheme storage proofs carry data chunks (as part of data access process) and hence are not *chunk-free*, making it subject to efficiency and security issues for on-chain auditing discussed before. Since the proposed index translation design itself represents the best one in the literature, we will use it to devise a strawman scheme for comparison (see Section 6.1).

As will be shown by our evaluation, an $O(n)$ auditor state, even with the optimal design in our OAD protocol, is still undesirable for on-chain auditing as it will greatly limit the capacity of real-world DS networks.

3 Overview

This section gives an overview of the on-chain auditing architecture. We present our main auditing protocols in Section 4 and data abstraction layer in Section 5.

3.1 System Model

A DS system is built with multiple layers. Architecture wise, the data auditing layer locates between and glues together the base storage layer and the upper incentive layer. It interacts with the underlying storage systems and provides necessary support, i.e., the evidences that outsourced data remains intact, to the correct functioning of incentive mechanisms. As shown in Fig. 1, we consider three parties in the system: the *data owner* outsources data files and auxiliary authentication information to the *storage node*, and the latter periodically submits storage proofs to the *blockchain* for auditing. The data owner can make arbitrary changes to the files by communicating the updates to the other two parties.

A data owner can distribute files, which are usually erasure coded, to multiple storage nodes to increase data availability. If data corruption is detected at a node, the DS will initiate a recovery procedure to restore the data to a healthy level. This requires some threshold number of nodes to operate normally. The requirement is pertinent to the DS but not the auditing design. In our design storage nodes are audited individually, meaning that a data owner will sign separate contracts with different storage nodes. This is reasonable as nodes in decentralized networks often do not work in a concerted way, and they differ in capacity and promised quality of services. Hereafter, we focus on a single pair of data owner and storage node.

Blockchain model. We use a public blockchain as a decentralized infrastructure as it is, and we do not consider designing a consensus protocol based on PoS as in some existing works. Everything stored on the blockchain is publicly accessible and tamper-resistant. All parties have their public keys recorded on the blockchain, and the messages between them are authenticated, i.e., signed with the associated blockchain signature scheme. Also, we assume the messages (i.e., transactions) the parties send to the blockchain will be delivered within certain time limit.

The on-chain auditing functionality is implemented as a smart contract that we call *auditor contract*. It contains functions for storage contract setup, data auditing, update, etc. A challenge function for auditing is scheduled to run automatically at predefined intervals. Other functions are invoked when receiving corresponding transaction messages from the data owner and storage node. The off-chain parties get notified of the execution status of functions by actively listening to blockchain events. Auditing requires the contract to use secure random numbers, which can be drawn from oracle contracts (e.g., RANDAO) or decentralized randomness beacons [43].

The auditor contract should define interfaces to handle disputes between off-chain parties and to interact with the DS's incentive mechanism, which could be implemented in other contracts. We refrain from discussing these functions in details and focus on the core auditing design in this paper.

Setup mode. In our architecture, the owner sets up the auditing protocols on its data locally. Such *private setup* is considered by default in the PoS literature. The data being owned does not imply that it is private to the owner: it may comprise widely spread or publicly accessible files, such as open-source code repositories or internet archives.[4] In other words, the private setup is meaningful to the owner regardless of the nature of data.

Some recent DS projects like Filecoin consider a *public setup* where the PoS is set up by the storage node. This makes sense for public files. But the authenticity of the owner's private files is not guaranteed, as they may already be altered by the storage node before the setup. In this case, the owner needs to use an additional layer of security measure to safeguard data integrity, in addition to the auditing mechanism provided by the DS.

4. In fact, deduplicating common files across users is a key cost saver to cloud storage providers [34]. Cross-user data deduplication in DS is an interesting topic on its own.

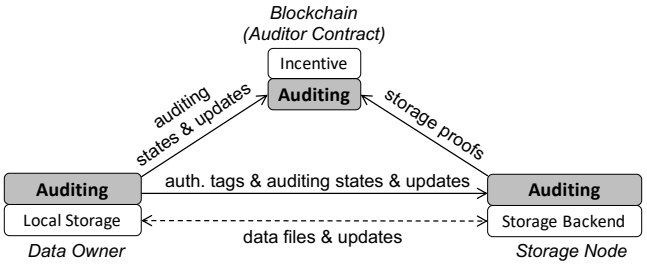


Fig. 1: Our system model with three parties. The data auditing layer builds on the underlying storage layer and serves for the upper incentive layer in the DS architecture. The arrows indicate main data flows. The dashed line indicates normal data retrieval process.

We emphasize this distinction because it leads to different security implications, especially when one wants to obtain integrity assurance for multiple replicas of the same data (aka Proof-of-Replication). In public setup the storage node can cheat data owners by storing less data than required, by means of compressing data, creating Sybil identities or colluding with others [2]. These problems can be trivially solved in private setup, as the owner itself can encrypt data replicas with different keys to make them independent to each other [16].

3.2 Threat Assumptions

We assume the blockchain platform is trusted and auditor contract is correctly implemented. We do not consider attacks targeting the blockchain itself [38]. The data owner and storage node distrust each other. Data corruption at the storage node may happen accidentally as a result of software/hardware failure, human error, or attacks from parties external to the system. The storage node may also intentionally discard portions of the hosted data to reclaim space for more storage contracts. Denial-of-service is out of scope, e.g., the node refuses to accept setup or updates from the data owner. In line with the open economy of DS, we consider a malicious data owner as well. It may falsely accuse the storage node for potential compensation enforced by the contract. Finally, we assume the communication channels between the parties are authenticated.

Previous research [46] shows that in the setting of public storage auditing, a third-party auditor can recover the original chunks from the proofs by repeatedly challenging them. Whether this constitutes a serious concern in the context of DS is debatable, as data stored there is almost always encrypted by default. Nonetheless, thanks to the generality of our design, we can readily use an existing construction [20] to instantiate the underlying PoS scheme, making the storage proofs zero-knowledge and hence preventing on-chain data recovery.

3.3 Preliminaries

Common notations. For simplicity, we denote the data owner as OW, the storage node as SN, and the auditing contract as AC. The concatenation of two strings a and b is $a\|b$. The notation $y := \mathsf{expr}$ assigns the value of an expression expr to y. The symbol $\perp$ is a special null value.

A tuple is denoted by $(x_1, x_2, \dots)$. We write $(y, \_)$ to access the value y from a tuple and discard others. The notation $y \leftarrow f(x)$ means assigning the output of an algorithm f over input x to y; for randomized algorithms we use $y \leftarrow\!\!\$\ f(x)$. We access an algorithm B of a scheme/data structure A by $\mathsf{A}.\mathsf{B}$. We use the compact set notation $\{a_i\}_n := \{a_1, \dots, a_n\}$ and similarly $\{a_i\}_{i \in \mathcal{I}}$, where the subscripts can vary.

Blockchain-related notations. All messages between off-chain parties and AC are signed with respective blockchain key pairs. When we need to make this explicit, we denote such a signature from party X by σ_{X}. A message sent to AC in the form $(\mathsf{fun}, \dots)$ means to call a smart contract function with corresponding parameters.

Plain array. As the most basic data structure, a plain array consists of a sequence of fixed-size data entries. The entries are identified the natural index $1, 2, \dots, n$. We write $\mathsf{A}[i]$ to access the i-th entry of an array A. Unlike other complex data structures, a plain array has almost no space overhead. That is, storing n data items of m bytes will most likely occupy just $n*m$-byte space (omitting low-level architecture features). We denote the number of cells in an array by $|\mathsf{A}|$.

Aggregate signature. An aggregate signature scheme [13] allows the signatures on multiple distinct messages from different signers to be combined into and verified as a single signature. Here, we only consider the simple case where all signatures are issued under a *single* key pair. The scheme ASig consists of a tuple of four algorithms: $(sk, pk) \leftarrow\!\!\$\ \mathsf{KeyGen}(\lambda)$, $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(sk, m)$, $\sigma \leftarrow \mathsf{Aggr}(pk, \{(m_i, \sigma_i)\}_n)$, as well as $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{Verify}(pk, \{m_i\}_n, \sigma))$.

4 On-chain Auditing of Dynamic Data

This section starts with an analysis of the data dynamism problem. We then introduce our basic dynamic auditing protocol that achieves an optimal $4n$-bytes on-chain index state (Section 4.2), followed by our improved auditing protocol that reduces the state to $10n/m$-bytes without losing practicality (Section 4.3). Finally, we show an efficient instantiation of our protocols for realistic use (Section 4.4).

4.1 Problem Analysis

To better understand the nature of data dynamism in storage auditing, we first introduce a schematic definition of PoS with explicit index information. The definition is adapted from the classic constructions based on homomorphic linear authenticators (HLA) [5], [7]. Let $\mathcal{F} := \{f_i\}_{i=1}^n$ be the set of chunks of an input file. The logical index of a chunk f_i is simply i. The chunks are naturally ordered by their logical indexes. A (public) PoS scheme consists of the following algorithms.

- $\mathsf{KeyGen}()$ generates a key pair (sk, pk).
- $\mathsf{GenTag}(sk, f, x)$ outputs an authentication tag π over a chunk $f \in \mathcal{F}$ and some index information x.
- $\mathsf{VeriTag}(pk, f, x, \pi)$ checks if the tag verifies against the chunk and the supplied index information.
- $\mathsf{Chal}(c)$ outputs a challenge $\mathcal{Q}$, which contains a set $\mathcal{I}$ of c random logical indexes and the same number of random weights drawn from proper domains.[5]
- $\mathsf{Prove}(pk, \mathcal{F}, \mathcal{T}, \mathcal{Q})$ outputs a proof Prf on input $\mathcal{F}$, the set of tags $\mathcal{T}$, and the challenge $\mathcal{Q}$. The size of Prf is typically independent of n and c.
- $\mathsf{Verify}(pk, \mathcal{Q}, \mathsf{Prf}, \mathcal{X})$ checks if the proof Prf verifies against $\mathcal{Q}$ and an index information set $\mathcal{X}$.

An authentication tag cryptographic binds a chunk and its index. It should be computationally impossible to find f, x, f', x' such that $x \neq x' \vee f \neq f'$ but $\mathsf{VeriTag}(pk, f, x, \mathsf{GenTag}(sk, f', x')) = \mathsf{true}$. Constructions of the scheme can produce compact storage proof, i.e., Prf is mainly comprised of chunks and tags in *aggregated forms*.

A typical system flow using the scheme works as follows. A client (i.e., OW) runs a setup procedure, generating a key pair and computing authentication tags for the input file. It then sends the chunks and tags to a storage server (i.e., SN) and deletes the file locally. After that, an auditor (i.e., AC) can periodically challenge the server by asking for storage proofs for random subsets of the chunks.

In the case of static data, one simply uses logical index for all algorithms, i.e., $x = i$ for chunk f_i and $\mathcal{X} = \mathcal{I}$. This enables the auditor, while possessing no information other than the public key, to verify that the returned proof indeed corresponds to the challenged chunks. A proof for any other combination of chunks will be falsified because of the cryptographic binding between chunks and indexes.

Using logical index however poses a problem on dynamic data. Suppose the client wants to insert a new chunk f' at position k. Then, in addition to computing the tag π'_k from f' and k, it has to retrieve chunks $f_k, f_{k+1}, \dots$ from the server and compute new tags for them with shifted logical index $k+1, k+2, \dots$, as the old tags become no longer valid. The case of deletion is similar. In other words, a single update may be as inefficient as re-running the expensive setup procedure. The resulting linear computation and communication cost is prohibitive and unacceptable.

For efficient support of data dynamism, Erway et al. [21] suggest that tags should not bind chunks with any explicit index information. To allow sublinear update costs, they propose to use some ADS to authenticate homomorphic tags over chunks. The tags themselves are orderless, and the order of chunks is indirectly represented as the structural rank information of the tree-like ADS. A storage proof includes for each challenged chunk an ADS witness (i.e., a tree path) with ranks, from which the auditor can reconstruct logical indexes needed for verification. As discussed before (Section 2.2), the large storage proofs produced by such design hinders its applicability to on-chain auditing.

Our approach. We seek to resolve the above issue of data dynamism while retaining the PoS scheme's efficiency. Our key idea is to bind chunks with alternative index information that is *insensitive* to data dynamism. Ideally, the update of a chunk should only affect its own tag. We call such an index *pseudo index*, denoted by p_i for chunk f_i.

5. The challenge $\mathcal{Q}$ can be compressed to a constant size independent of c, i.e,. only two PRF seeds, with standard techniques [5], [39]. We keep the current definition for simplicity.

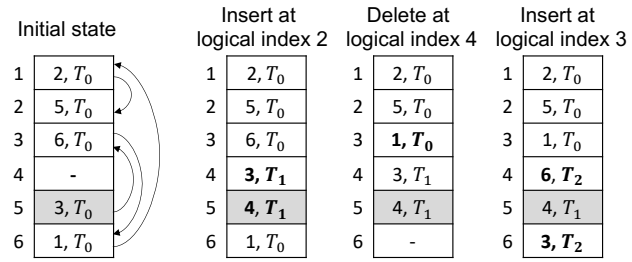


Fig. 2: Illustration of PIL implemented with a plain array. Each entry contains a pointer and possibly other small metadata (here, a timestamp). The shaded cell is the PIL's head. The updated data is marked in bold font. For the first and third update operation, only one new tag need to be computed with pseudo index 4 and 6, respectively.

For the proposed design to work correctly, there must be a one-to-one correspondence between logical indexes and pseudo indexes. It is impossible to define a fixed function to describe the mapping between the two types of indexes, as in general data insertion and deletion can happen at arbitrary position for arbitrary times. As a result, the mapping should be stored in some data structure, introducing a non-trivial state to be kept by the parties involved in the PoS scheme. It allows the parties to obtain necessary index information for tag computation, proof generation and verification, as well as update operations.

This approach can transform any static PoS scheme into a dynamic one. The security guarantee of data integrity or retrievability remains the same, as long as all pseudo indexes in use any point in time are distinct and that the (public) auditor always maintains the correct state. Security notions of dynamic PoS have been formalized by previous work (e.g., [24], [53]). Since our focus is on designs for efficiency, we refrain from providing a verbose definition of the already well-established security framework in this paper.

While similar ideas of index translation have been studied before in the context of cloud storage, the unique design space of on-chain auditing with more stringent performance requirements has never been explored before. Our primary design goal here is to minimize on-chain costs of managing the state, and we accomplish this by developing a set of novel succinct data structures and optimization techniques. In our subsequent protocol design, we use the static PoS scheme as a black box and specify inputs to its algorithms with concrete values. At the end of this section (§4.4), we will give an instantiation of PoS that facilitates our generic dynamic on-auditing protocols to achieve optimal efficiency.

4.2 Our Basic Design

To begin with, we observe that as long as the index mapping is stored in the explicit pair form (i, p_i), insertion and deletion will always have $O(n)$ time complexity regardless of the data structure used. The reason is that we have to update the pairs for all logical indexes greater than the inserted/deleted position. A similar situation arises when storing pseudo indexes in a plain array where logical indexes equals array indexes (as in [32], [51], [53]). While the concrete cost of $O(n)$ index mapping update is not as significant as tag computation and is acceptable for off-chain parties, it is unreasonable for an on-chain auditor.

The state IS includes
- A : a plain array to realize a PIL. Each entry stores the address (pseudo index) of next node in the PIL and the timestamp of current node;
- C : the capacity of A;
- head : a pointer to the PIL's starting node;
- free : a pointer to a list of allocatable address.

$\mathsf{Init}(n, t)$: // initialize IS with length n at time t
- Set $C := n, \mathsf{head} := 1,$ and $\mathsf{free} := \perp$
- Set $A[i] := (i + 1, t), \forall i \in [1, n]$

$\mathsf{GetPI}(i)$ outputs (p_i, t_i) : // translate logical index i
- Access $(p_2, t_1) := \mathsf{A}[\mathsf{head}], (p_3, t_2) := \mathsf{A}[p_2], \ldots,$ until obtaining $(p_i, t_{i-1}) := \mathsf{A}[p_{i-1}]$ and $(p_{i+1}, t_i) := \mathsf{A}[p_i]$

$\mathsf{Mdfy}(i, p_i, t)$: // update at logical index i and time t
- Set $(\tilde{p}, \_) := \mathsf{A}[p_i]$ and then $\mathsf{A}[p_i] := (\tilde{p}, t)$

$\mathsf{Isrt}(i, p_{i-1}, t)$: // insert at logical index i and time t
- If $\mathsf{free} = \perp$ then set $C := C + 1$ and $\mathsf{free} := C$
- Set $\mathsf{freed} := \mathsf{free}, (\mathsf{free}_{nxt}, \_) := \mathsf{A}[\mathsf{free}], \mathsf{free} := \mathsf{free}_{nxt}$
- Set $(\tilde{p}, \tilde{t}) := \mathsf{A}[p_{i-1}]$
- Set $\mathsf{A}[\mathsf{freed}] := (\tilde{p}, t)$ and $\mathsf{A}[p_{i-1}] := (\mathsf{freed}, \tilde{t})$

$\mathsf{Dlte}(i, p_{i-1})$: // delete at logical index i
- Set $(p_i, t_{i-1}) := \mathsf{A}[p_{i-1}]$ and $(\tilde{p}, \perp) := \mathsf{A}[p_i]$
- Set $\mathsf{A}[p_{i-1}] := (\tilde{p}, t_{i-1})$
- Set $\mathsf{A}[p_i] := (\mathsf{free}, \perp)$ and $\mathsf{free} = p_i$

Fig. 3: Our basic index state design.

We find that the index mapping can be elegantly encoded by a linked list. Suppose each node of the list stores a data chunk in correct order. Then, the logical index of a chunk is simply its position on the list, and the pseudo index can be set as the address of its node. The index mapping is automatically maintained by the linked list during updates. The advantage of this design is that insertion or deletion at a given node has constant complexity. Note that the linked list does not need to store the actual chunks, so we can realize it using a plain array storing only pointers (and other small metadata). We call the resulting succinct data structure pseudo index linked list (PIL) and give an illustration in Fig. 2. It serves as the basis for our subsequent designs.

Keeping track of only index information is insufficient to securely audit dynamic data. The same pseudo index can be assigned to different chunks over time. This creates the possibility of a relay attack where SN forges proofs using out-dated, invalid chunks with matching indexes. To ensure freshness, we associate the latest update time to each chunk and store the timstamps as part of the state. We denote the resulting state by IS and describe its full details in Fig. 3.

IS is essentially a plain array of two integers, each of which can be represented with 4 bytes for practical use. After the initialization, insertion will increase the capacity of IS if the array is full, and deletion will free addresses and make them allocatable. To avoid space waste, we reorganize array entries if the PIL's length decreases to a certain threshold, e.g., half of the capacity. The cost of re-

Setup :

1. OW computes on input $\mathcal{F} := \{f_i\}_{i=1}^{n}, t := \mathsf{CrtTime}$:
   $(pk, sk) \leftarrow\!\!\$\ \mathsf{PoS.KeyGen}(\lambda)$;
   $\mathsf{IS.Init}(n, t)$;
   $\mathsf{A_{ac}}[i] := t, \forall i \in [1, n]$;
   $\mathcal{T} := \{\pi_i \leftarrow \mathsf{PoS.GenTag}(sk, f_i, (i+1)\|t)\}_n$;
   and sends $(pk, \mathcal{F}, \mathsf{IS}, \mathsf{A_{ac}}, \mathcal{T})$ to SN
2. SN verifies all tags by $\mathsf{PoS.VeriTag}$, checks that IS and $\mathsf{A_{ac}}$ are consistent, and sends its signature σ_{SN} on $(pk, \mathsf{A_{ac}})$ back to OW.
3. OW verifies σ_{SN} and sends $(\mathsf{Setup}, pk, \mathsf{A_{ac}}, \sigma_{\mathrm{SN}})$ to AC.
4. AC verifies σ_{SN} and stores pk and $\mathsf{A_{ac}}$ as its state. It then emits a setup completion event.
5. OW deletes $\mathcal{F}$ and $\mathcal{T}$ while keeping sk and IS locally.

Audit :

1. AC generates a challenge $\mathcal{Q} \leftarrow \mathsf{PoS.Chal}(c)$, samples a random $\mathcal{P} \subset [1, |\mathsf{A_{ac}}|]$ of size c s.t. $\mathsf{A_{ac}}[p] \neq \perp, \forall p \in \mathcal{P}$, and sets $\mathcal{X} := \{(p, \mathsf{A_{ac}}[p])\}_{p\in\mathcal{P}}$. It then sets a deadline $\mathsf{Ddl_a}$ and emits a challenge creation event with $(\mathcal{Q}, \mathcal{P})$.
2. SN computes $\mathcal{I}' := \{i : (p, \_) = \mathsf{GetPI}(i)\}_{p\in\mathcal{P}}$ with one pass through IS, and replaces $\mathcal{I}$ in $\mathcal{Q}$ with $\mathcal{I}'$. It then sends a proof $\mathsf{Prf} \leftarrow \mathsf{PoS.Prove}(pk, \mathcal{F}, \mathcal{T}, \mathcal{Q})$ to AC.
3. AC checks $\mathsf{CrtTime} \leq \mathsf{Ddl_a}$ and $\mathsf{PoS.Verify}(pk, \mathcal{Q}, \mathsf{Prf}, \mathcal{X})$.

Update :

Let $\mathsf{Upd} \in \{(\text{“Mdfy”}, i, t), (\text{“Isrt”}, i, t), (\text{“Dlte”}, i, t)\}$ be an update message and f' a new chunk (empty for deletion).

1. On input (Upd, f'), OW computes a new tag π' for f'. It then run corresponding algorithms in Fig. 3 to update IS. It also prepares the update info needed by AC, $\mathsf{Upd_{ac}}$, which contains at most two pairs of array index and timestamp. Next, OW generates a signature σ_{OW} on (Upd, f', π'). Finally, it sends $(\mathsf{ProposeU}, \mathsf{Upd_{ac}})$ to AC and $(\mathsf{Upd}, f', \pi', \sigma_{\mathrm{OW}})$ to SN.
2. AC records $\mathsf{Upd_{ac}}$ and sets a deadline $\mathsf{Ddl_u}$. It will reject any new update proposal until step 4 is completed.
3. SN updates its own copy of IS, and verifies π' against f' by $\mathsf{PoS.VeriTag}$ and validates $\mathsf{Upd_{ac}}$ on AC. If they are all correct, SN commits local changes and sends $\mathsf{ConfirmU}$ to AC. Otherwise, it reverts local changes and optionally sends $(\mathsf{AbortU}, f', \pi', \sigma_{\mathrm{OW}})$ to AC.
4. AC discards $\mathsf{Upd_{ac}}$ if no message is received before $\mathsf{Ddl_u}$. Otherwise, it updates $\mathsf{A_{ac}}$ with $\mathsf{Upd_{ac}}$ upon receiving $\mathsf{ConfirmU}$, or report a dispute upon receiving AbortU.

Fig. 4: Our first on-chain auditing protocol OAD.

organization is amortized over updates. The lookup algorithm GetPI has time complexity $O(n)$ to traverse the PIL and all three update algorithms run in trivial constant time.

A straightforward way to enable data dynamism is synchronizing IS across the three protocol parties. The costs of maintaining IS should place no burden on the two off-chain parties. This is however suboptimal to AC, as it needs to traverse through the IS to obtain pseudo indexes corresponding to the challenged logical indexes for proof verification. Next, we show how to optimize both the storage and computation costs of AC.

Optimizing on-chain auditor. Note that instead of logical indexes, the auditor can sample the chunks directly with their pseudo indexes, or equivalently the array indexes of IS's A. The storage node is able to map the pseudo indexes in a challenge back to logical indexes and generate the proof as usual. Since there is no need for the auditor to do index translation, now it can just keep an array of timestamps that are consistent with the IS of the other two parties. We denote such a minimal array by $\mathsf{A_{ac}}$. The auditor can easily update $\mathsf{A_{ac}}$ with the pseudo (array) indexes and timestamps sent from the client. Deletion will create empty entries in $\mathsf{A_{ac}}$. We store in such entries with a special value 0. This allows the auditor to discern invalid pseudo indexes and make ensure that all sampled ones in a challenge indeed correspond to existing chunks. To sum up, with the optimization technique we can half the on-chain state size and make proof verification on dynamic data as efficient as the case of static data.

Our first generic protocol. Figure 4 presents OAD, our first on-chain auditing protocol for dynamic data. It consists of three phases. The extra operations we introduce to support dynamic data centers around index state manipulation and mutual data verification between parties. Initially, the pseudo index for logical index i is just $i + 1$. This simple relation will be gradually changed by subsequent data update operations. To perform an update operation, OW first proposes it to AC; after examining the update information, SN will either confirm or abort the proposed operation. The on-chain update cost is $O(1)$: AC needs to write at most two $\mathsf{A_{ac}}$ entries at known positions (one for modification and two for insertion/deletion).

Sometimes multiple update operations can happen within a short period of time, say when a large commit is pushed to a repository hosted in the DS. In this case, we can handle all of them in a single round to save on-chain cost, in particular the fixed, non-trivial blockchain transaction overhead. Now the $\mathsf{Upd_{ac}}$ will aggregate the information for all updates and be sent to AC in a single transaction.

Security analysis. First of all, OAD inherits data integrity assurance against a malicious SN from the underlying PoS scheme. AC always keeps the latest, authentic state information received from OW that is free from tampering. Over the time, no two legitimately generated tags will ever be binded to the same pair of pseudo index and timestamp. Thus, if a proof is verified, it must be either faithfully created from the latest challenged chunks with matching pseudo indexes or forged by SN. The latter case is impossible given the computational hardness assumptions relied on by the PoS.

Second, we consider a dishonest OW that attempts to frame SN. During the setup phase, only after validating all received data will SN sign the setup transaction. OW cannot persuade AC to accept a different auditor state without a valid signature from SN. Similarly, in subsequent update phase SN will only accept update that is consistent with what is proposed to AC. SN can further choose to submit evidence, i.e., OW's signature on the proposed update data, to AC for potential dispute resolution. In conclusion, OW is free from being wrongfully convicted, because it always keeps the latest, valid tags and state to pass data audits.

The large IS^+ contains, with reference to Fig. 3:
$\mathsf{IS}_1, \mathsf{IS}_2, \ldots$: a list of ISes that share the *same* A, C and free. Each IS_i manages a segment of indexes with its own head_i and a similar tail_i pointer. m is the maximum segment size and Δ is a load balancing parameter.
The small IS^- contains
A_* : A plain array to realize PIL for segment information. Similar to IS, each entry here contains a pointer ρ, a timestamp τ, and the current segment size η.
$\mathsf{C}_*, \mathsf{head}_*, \mathsf{free}_*$: Similar to that in IS

// Algorithms for IS^-
$\mathsf{GetSeg}(i)$ outputs (k, ζ_i) :
Scan A_* until hit a k s.t. $\sum_{j=1}^{k} \eta_j < i \leq \sum_{j=1}^{k+1} \eta_j$
Set $l := i - \sum_{j=1}^{k} \eta_j$ and $\zeta_i := (l, \rho_k, \tau_k, \eta_k)$
// These algorithms work in the same way as those of IS and thus are skipped for savings of space.
$\mathsf{Init}^-(n, t, \eta), \mathsf{Mdfy}^-(k, \rho, \tau, \eta), \mathsf{Isrt}^-(k, \rho, \tau, \eta), \mathsf{Dlte}^-(k, \rho)$

// Algorithms for IS^+
$\mathsf{Init}^+(n, t)$:
Run $\mathsf{IS}_k.\mathsf{Init}(m, t)$, for $k = 1, 2, \ldots, n/m$
Run $\mathsf{IS}^-.\mathsf{Init}^-(n/m, t, m)$ // $\mathsf{A}_*[j] = (j+1, t, m)$
$\mathsf{GetPI}^+(i)$ outputs (p_i, t_i) :
$(k, \zeta_i) \leftarrow \mathsf{IS}^-.\mathsf{GetSeg}(i)$
$(p_i, t_i) \leftarrow \mathsf{IS}_k.\mathsf{GetPI}(l)$ // l is local logical index in IS_k
$\mathsf{Mdfy}^+(i, t)$:
$(k, (l, \rho_k, \tau_k, \eta_k)) \leftarrow \mathsf{IS}^-.\mathsf{GetSeg}(i)$
$(p_{i-1}, \_) \leftarrow \mathsf{IS}_k.\mathsf{GetPI}(l-1)$
// ρ_{k-1} is obtained when running $\mathsf{GetSeg}(i)$
Run $\mathsf{IS}^-.\mathsf{Mdfy}^-(k, \rho_{k-1}, t, \eta_k)$ and $\mathsf{IS}_k.\mathsf{Mdfy}(l, p_{i-1}, t)$
$\mathsf{Isrt}^+(i, t)$:
Obtain $k, (l, \rho_k, \tau_k, \eta_k), p_{i-1}, \rho_{k-1}$ as above
Run $\mathsf{IS}_k.\mathsf{Isrt}(l, p_{i-1}, t)$
Set $\eta_k := \eta_k + 1$ and call $\mathsf{IS}^-.\mathsf{Mdfy}^-(k, \rho_{k-1}, t, \eta_k)$
If $\eta_k = (m+1)$ then // re-balancing on overflow
// The current IS_{k+1} is already almost full
If $\eta_{k+1} > \Delta$ then // η_{k+1} is from $\mathsf{A}_*[\rho_k]$
Create a new, empty segment by $\mathsf{IS}_{k+1}.\mathsf{Init}(0, t)$
Run $\mathsf{IS}^-.\mathsf{Isrt}(k+1, \rho_k, t, 0)$
$\overline{m} := (\eta_k + \eta_{k+1})/2$
Obtain $(p', t') \leftarrow \mathsf{IS}_k.\mathsf{GetPI}(\overline{m})$ and $(p'', t'') := \mathsf{A}[p']$
Set $\mathsf{tail}_k := p'$ and $\mathsf{head}_{k+1} := p''$
Run $\mathsf{IS}^-.\mathsf{Mdfy}^-(k, \rho_{k-1}, t, \overline{K})$
Run $\mathsf{IS}^-.\mathsf{Mdfy}^-(k+1, \rho_k, t, \overline{K}+1)$
// $\mathsf{Dlte}^+$is similar to $\mathsf{Isrt}^+$and skipped for space limit.

Fig. 5: Our improved index state design.

4.3 Our Improved Design

The $O(n)$ on-chain state of OAD limits its applicability to realistic datasets of large volumes. We seek to compress the state as much as possible. As a starting point, consider a conceptual design where AC does not store the index state but instead rely on SN to obtain the pairs $\{(p_i, t_i)\}_{i \in \mathcal{I}}$.

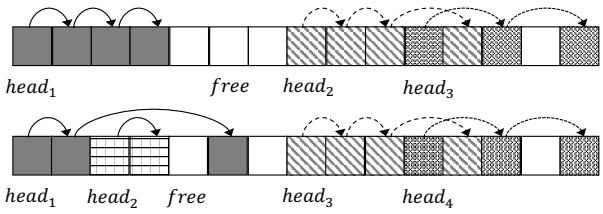


Fig. 6: The illustration of segmented index state with $m = 4$ and three initial PILs (above). After an insertion at global logical index 3, the first PIL is split into two (below). The pointers for the shared free list are not shown for clarity.

Now for security we must authenticate each pair with its logical index binded, i.e., the tuple (i, p_i, t_i); if the authenticator allow aggregate verification, the overall storage proof should remain short. However, this design brings us back to the dilemma that insertion or deletion incurs $O(n)$ costs to re-authenticate the tuples. Of course, the good news is authenticating small index tuples is much cheaper than recomputing tags.

The discussion above reveals a fundamental trade-off between on-chain state and off-chain update efficiency; our OAD optimizes the latter while the conceptual design suggested above optimizes the former. Neither of the two extremes is desirable, and we strive to achieve a sweet spot in the middle. Our solution is built around a novel index space segmentation technique. Let m be the maximum segment size, which is a tunable parameter. Our technique allows for a m-factor reduction of $O(n)$ on-chain index state while maintaining $O(m)$ off-chain update costs for index state authentication (still, only one tag needs to be recomputed for each update). This simultaneously brings all performance metrics of the resulting auditing protocol into realistic ranges.

Index space segmentation. In OAD we have a single PIL to manage the index state, and now we need a separate PIL for each segment. A segment manages the index mapping for a consecutive sequence of logical chunk indexes, and the segments themselves are naturally ordered. A *global* logical chunk index can be uniquely represented in the form of a segment index and a *local* logical chunk index within the segment. By binding the latter form with pseudo index pair (p_i, t_i) in the tuple for authentication, we can localize the cascade effect of inserting and deleting a chunk to its own segment. To be able to verify the authenticated index tuple in a proof, AC must keep the current sizes of all segments. To thwart replay attack, it should also keep track of timestamps for latest updates taking placing in the segments. In other words, AC's index state now becomes metadata about segments. While the overall idea of segmentation might look intuitive at first glance, non-trivial technicalities are yet to be handled in order to attain the expected efficiency.

First, updates of data chunks may lead to the insertion and deletion of segments with a fixed maximum size. This will cause the shift of segment indexes and consequently $O(n)$ costs to re-authenticate the affected index tuples. To this end, we apply the concept of PIL again at the segment level. Instead of *logical segment index*, we use *pseudo segment index* in the tuple for authentication. The insertion and deletion of a segment will not affect others' pseudo segment

indexes. This design produces a two-layer structure: one PIL for segment index, and a set of PILs for chunk index.

Besides, updates may cause a skewed segment size distribution and hence performance degradation. Too many underutilized segments will produce an on-chain state larger than necessary. We address this issue by rearranging chunk-level PILs whenever necessary. Specifically, insertion into a full segment will trigger relocation of entries between it and its neighbor, which is either an existing segment that is not yet full (up to a load balancing parameter) or a newly created empty one. Similarly, when the size of a segment decrease to a certain threshold, we will either balance its load with its neighbor or merge the two segments into a single one. With such design we can keep all segments always in a highly utilized and balanced status.

Another subtle issue is that relocating entries between segments can force an excessive number of tags to be recomputed. The reason is that the pseudo index p_i of a chunk in one PIL will likely change when the chunk is "moved" to another PIL. We resolve this issue by having the PILs share a single address space or array. Consequently, load balancing two segments only involves updating their head and tail pointers with negligible cost. Figure 6 gives an illustration.

We describe our full design in Fig. 5. Segmentation results in two separate states: the IS^+ that consists of a set of ISes for chunk index mapping and the IS^- for segment information management. The two off-chain parties will store both states. Compared with the IS in our basic design, IS^+ introduces $2n/m$ extra pointers, the storage overhead of which is insignificant for a practical choice of m. AC only needs to store IS^-. It is essentially an array of three integers: a pseudo segment index (4 bytes), a timestamp (4 bytes), and a segment size (2 bytes, allowing for a maximum segment size of up to 65536). The concrete size of IS^- is thus roughly $10n/m$ bytes and can be made very small in practice, as will be shown in our evaluation.

Our second generic protocol. Figure 7 presents our second on-chain auditing protocol OAD^+. The major difference between it and OAD lies in how we handle index information. Here, AC relies on authenticated chunk index information sent from SN to verify PoS proofs. We use an aggregate signature scheme ASig to authenticate index tuples. This guarantees that the overall storage proof remains compact. During the audit phase, AC samples logical chunk indexes to challenge SN. The optimization technique introduced in Section 4.2 is not applicable here, because AC needs the order of segments and their sizes to derive the local logical chunk index for a sampled global logical chunk index. This means that AC needs to traverse through IS^- to obtain necessary information for proof verification. Nevertheless, the m-factor reduction in the number of array entries makes the cost of linear scan on IS^- marginal compared with other operations in proof verification, as confirmed by our experiment results. Similar to OAD, on-chain update of OAD^+ is extremely efficient as it involves running just an $O(1)$ update algorithm on IS^-.

Security analysis. Compared with OAD, the data integrity assurance of OAD^+ additionally depends on the authenticity of index tuples. AC always keeps the latest, authentic IS^- from OW, and the segment metadata ζ_i is unique for all global logical chunk index i over time. If the aggregate signature σ in the storage proof verifies, then by the security of the ASig scheme the pairs $\{(p_i, t_i)\}_{i\in\mathcal{I}}$ must correspond to the challenged $\{\zeta_i\}_{i\in\mathcal{I}}$ and hence the sampled chunks. The handling of a malicious OW is also similar to that in OAD. SN can always check the validity of signatures on index tuples and reject any data inconsistency to protect itself from being framed.

Setup :

1. OW computes on input $\mathcal{F} := \{f_i\}_{i=1}^n, t := \mathsf{CrtTime}$:
   $(pk, sk) \leftarrow_\$ \mathsf{PoS.KeyGen}(\lambda)$;
   $(ipk, isk) \leftarrow_\$ \mathsf{ASig.KeyGen}(\lambda)$;
   $\mathsf{IS}^+.\mathsf{Init}^+(n, t)$;
   One pass of IS^-, IS^+ to get $\{(p_i, t_i, \zeta_i)\}_n$
   $\mathcal{T} := \{\pi_i \leftarrow \mathsf{PoS.GenTag}(sk, f_i, p_i \| t_i)\}_n$;
   $\mathcal{E} := \{\sigma_i \leftarrow \mathsf{ASig.Sign}(isk, p_i \| t_i \| \zeta_i)\}_n$;
   and sends $(\mathcal{F}, pk, ipk, \mathcal{T}, \mathcal{E}, \mathsf{IS}^+, \mathsf{IS}^-)$ to SN.
2 – 5. Same as Fig. 4, except that: 1) SN also verifies $\mathcal{E}$, 2) σ_{SN} is computed on (pk, ipk, IS^-), and 3) AC stores (pk, ipk, IS^-) as its state.

Audit :

1. AC computes $\mathcal{Q} \leftarrow \mathsf{PoS.Chal}(c)$, sets a deadline $\mathsf{Ddl_a}$, and sends $\mathcal{Q}$ to SN.
2. SN computes, on input $\mathcal{Q}$ (which contains $\mathcal{I}$) :
   $\mathsf{Prf} \leftarrow \mathsf{PoS.Prove}(pk, \mathcal{F}, \mathcal{T}, \mathcal{Q})$;
   // One pass of IS^+ is sufficient to get all (p_i, t_i).
   $\mathcal{X} := \{(p_i, t_i) \leftarrow \mathsf{IS}^+.\mathsf{GetPI}^+(i)\}_{i\in\mathcal{I}}$;
   $\sigma := \mathsf{ASig.Aggr}(ipk, \{\sigma_i\}_{i\in\mathcal{I}})$;
   and sends $(\mathsf{Prf}, \mathcal{X}, \sigma)$ to AC.
   // One pass of IS^- is sufficient to get all ζ_i.
3. AC obtains $\{\zeta_i\}_{i\in\mathcal{I}}$ from $\mathsf{IS}^-.\mathsf{GetSeg}(i)$ and checks:
   $\mathsf{CrtTime} \le \mathsf{Ddl_a}$;
   $\mathsf{ASig.AggVeri}(ipk, \sigma, \{p_i \| t_i \| \zeta_i\}_{i\in\mathcal{I}}) \stackrel{?}{=} \mathsf{true}$;
   $\mathsf{PoS.Verify}(pk, \mathcal{Q}, \mathsf{Prf}, \mathcal{X}) \stackrel{?}{=} \mathsf{true}$.

Update :

1. OW, on input (Upd, f'), updates $\mathsf{IS}^+, \mathsf{IS}^-$ with respective algorithms in Fig. 5 and computes a new tag π'. It then computes $\mathcal{E}' := \{\sigma_j \leftarrow \mathsf{ASig.Sign}(isk, p_j \| t_j \| \zeta_j)\}_{\forall j\in\mathcal{J}}$ where $\mathcal{J}$ is the set of global logical indexes belonging to the affected IS_k (the rare case of two affected segments is omitted for simplicity). Now $\mathsf{Upd_{ac}} := (k, \rho, t, \eta)$ is the update info needed by AC. OW computes σ_{OW} and sends messages to AC and SN as in Fig. 4
2 – 4. Same as Fig. 4, except that SN also needs to verify the new signatures $\mathcal{E}'$ sent from OW

Fig. 7: Our second on-chain auditing protocol OAD^+.

4.4 Protocol Instantiation

While in principle our protocols can be instantiated with any PoS construction that complies with the scheme introduced in Section 4.1, a sensible choice should be made in order to achieve practical performance. It is tempting to set up

the auditing protocols with a large chunk size to keep the number of chunks, and hence the on-chain state, small. This is however not always desirable. In the classic public PoS constructions [5], [39], the storage proof contains an aggregated form of chunks and has a size slightly larger than chunk size; in other words, adopting them will disallow simultaneous optimization of on-chain state and proof size. Indeed, all previous dynamic protocols [32], [40], [45], [51], [53], if applied to on-chain auditing, would suffer from such a deficiency.

A desired PoS construction for instantiation should achieve constant-size proof that is independent of chunk size. One good candidate is the ones (e.g., [52]) that use polynomial commitment [33] to compress chunks. They generate efficiently verifiable proofs with only small group elements. Note that such design is still not perfect, as its public parameter now depends on chunk size, adding another factor to consider for the optimization of on-chain state; it is nonetheless the most suitable one we are aware of. Figure 8 summarizes the construction we use for instantiation. We do not claim contribution for the construction itself (which is already alluded to in previous work [19]), and the inclusion of it in the paper is for completeness purpose

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a non-degenerate and efficiently computable bilinear pairing, where $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T| = p$. Let g_1, g_2 be a generator of $\mathbb{G}_1, \mathbb{G}_2$, respectively. Also, use a full domain hash $H : \{0,1\}^* \to \mathbb{G}_1$. Each data chunk m_i consists of s elements $\{m_{ij}\}_{j=1}^{s} \in \mathbb{Z}_p^s$. The chunk can be represented by a polynomial $f_{\overrightarrow{m_i}}(x) := \sum_{j=0}^{s-1} m_{ij} x^j \in \mathbb{Z}_p$. Let $name$ be a file identifier chosen from a proper domain.

$\mathsf{KeyGen}() \to (sk, pk)$:

Pick two distinct random numbers $\alpha, \beta \leftarrow_{\$} \mathbb{Z}_p$

Compute $v := g_2^{\alpha}, z := g_2^{\alpha\beta}$, and $\{g_1^{\beta^j}\}_{j=0}^{s-1}$

Set $sk := (\alpha, \beta)$ and $pk := (v, z, \{g_1^{\beta^j}\}_{j=0}^{s-1}, g_2)$

$\mathsf{GenTag}(sk, m, x) \to \sigma$:

Set $\sigma := (H(name\|x) \cdot g_1^{f_{\overrightarrow{m}}(\beta)})^{\alpha}$

$\mathsf{VeriTag}(pk, m, x, \sigma) \to$ accept or reject :

Compute $g_m := g_1^{f_{\overrightarrow{m}}(\beta)}$ using $\{g_1^{\beta^j}\}_{j=0}^{s-1}$ in pk

Check if $e(\sigma, g_2) \stackrel{?}{=} e(H(name\|x) \cdot g_m, v)$

$\mathsf{Chal}(c) \to \mathcal{Q}$:

Sample a random subset $\mathcal{I} \subseteq [1, n]$

Set $\mathcal{Q} := \{(i, \nu_i \leftarrow_{\$} \mathbb{Z}_p)\}_{i \in \mathcal{I}} \cup \{r \in \mathbb{Z}_p\}$

$\mathsf{Prove}(pk, \mathcal{M} := \{m_i\}_{i=1}^{n}, \mathcal{E} := \{\sigma_i\}_{i=1}^{n}, \mathcal{Q}) \to \mathsf{Prf}$:

Aggregate the tags $\sigma := \prod_{(i,\nu_i) \in Q} \sigma_i^{\nu_i}$

Compute a polynomial $f_{\vec{A}}(x) \leftarrow \sum_{(i,\nu_i) \in Q} \nu_i \cdot f_{\overrightarrow{m_i}}(x)$

Compute $y \leftarrow f_{\vec{A}}(r) \in \mathbb{Z}_p$

Derive a polynomial $f_{\overrightarrow{w}}(x) \leftarrow (f_{\vec{A}}(x) - f_{\vec{A}}(r))/(x - r)$

Compute $\phi \leftarrow g_1^{f_{\overrightarrow{w}}(\beta)}$ using $\{g_1^{\beta^j}\}_{j=0}^{s-2}$ in pk

Set $\mathsf{Prf} := (\sigma, y, \phi)$

$\mathsf{Verify}(pk, \mathcal{Q}, \mathsf{Prf}, \mathcal{X}) \to$ accept or reject :

Compute $h := \prod_{(i,\nu_i) \in \mathcal{Q}, x_i \in \mathcal{X}} H(name\|x_i)^{\nu_i}$

Check with the public key (v, z) :

$e(h, v) \cdot e(\phi, z \cdot v^{-r}) \stackrel{?}{=} e(\sigma, g_2) \cdot e(g_1^{-y}, v)$

Fig. 8: An efficient PoS construction with constant-size proof for the instantiation of our dynamic auditing protocols.

5 Data Abstraction for Auditing

5.1 Rationale

In previous section we consider the data to audit as a single file that is divided into consecutive fixed-size chunks. This is the convention for most PoS designs. In practice, it is likely that the outsourced data consists of many files of different sizes. To this end, one may attempt to run a separate PoS setup for each of them and auditing them individually. This is undesirable from both security and efficiency perspective. If only a few files are covered in each audit, users cannot gain a strong and uniform assurance of their data. On the other hand, auditing most or all files at a time may overwhelm the computation and communication resources of the server and auditor. A more desirable approach is to consolidate all files so that they can be protected by the auditing mechanism as a whole.

Ideally, the consolidation should not depend on how files are organized by the underlying storage system. Some previous solutions [21], [42] are built on and tightly coupled with the OS file system, which structures files into a hierarchical directory tree. Here, chunks for auditing are naturally formed from the ordered leaf nodes at the bottom level of a directory tree. Yet, various storage alternatives exist in practice and especially in DS: key-value stores [17], object storages (FileCoin [2] and Storj [**?**] implement their own), databases [1], customized designs [3], etc. It may be difficult or even impossible to migrate an ad hoc solution for one storage system to another.

The need for handling data dynamics further complicates the issues. Adding or deleting files or changing their sizes may force many chunks to be updated with a cascade effect. Frequent data relocation may also cause severe fragmentation issue, resulting in many underused chunks and performance impact. A desirable design should confine changes of chunks to themselves and maintain all chunks in a balanced and highly utilized status.

Despite these complicated factors, it turns out what is needed by auditing is quite simple: just a *logical view* of the data as a sequence of ordered chunks. It does not matter to the auditing protocol how the data is organized and stored by the underlying storage system. By providing a proper data abstraction layer as illustrated in Fig. 9, we can decouple auditing from data storage and make our auditing protocols deployable on arbitrary storage systems. As another benefit, we find that in operation the small data abstraction layer can entirely reside in main memory. This allows more accurate profiling of auditing protocols and storage systems for performance optimization.

5.2 Lightweight Audit Chunk

The abstraction centers around what we call *lightweight audit chunk*, LAC for short. A LAC stores *imaginary* pieces of data. Specifically, it keeps an ordered list of metadata triplets $(\mathsf{id}, \mathsf{off}, \mathsf{len})$, where id is the identifier to a data object (i.e., a file), off is an offset from the beginning of the object in raw format, and len is the amount of data of the object starting

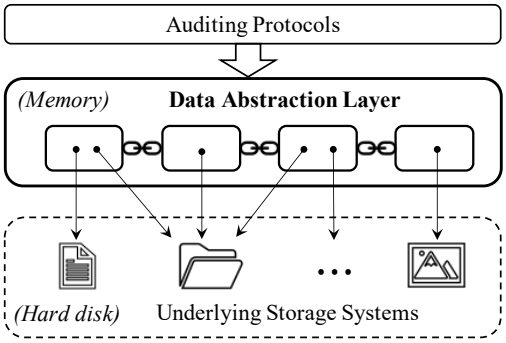


Fig. 9: Illustration of the data abstraction layer that hides the low-level details of data storages from auditing protocols.

from off. The triplets represent the objects that logically exist in the LAC, regardless of how they are stored underneath. With the triplets, we can retrieve the actual data by invoking interfaces provided by the underlying storage system.

The capacity of a LAC is the maximum amount of imaginary data it can hold, and its size equals the sum of len of all triplets it currently stores. A LAC is not necessarily full, i.e., its size may be less than or equal to its capacity. An object can cross multiple LACs. We always allocate an object in a minimally necessary number of consecutive LACs. This reduces data fragmentation and management overhead, as the total number of triplets is minimized.

Given a set of objects to be set up for auditing, we construct a linked list of LACs with fixed capacity to provide the desirable logical view of data. The LACs represent the chunks used by a PoS scheme (see Section 4.1). When computing an authentication tag, we construct the input chunk by concatenating the actual data retrieved from all triplets in corresponding LAC.

LACs can be freely modified, inserted, or deleted to serve for the auditing protocols. Updating a LAC involves manipulating the metadata triplets but not the real data and hence is very efficient. Yet, we need to maintain all LACs in a balanced status to ensure efficiency. The mechanism is similar to what we use for index segmentation. An insertion into a full LAC will cause the creation of a new chunk, or a data split between it and its neighbors if there is available space; in either case, we require the data of the affected adjacent chunks to be evenly distributed. Likewise, the LAC affected by a deletion may receive data from its neighbors to keep the load balance, or it may just merge with them depending on their utilization. By doing so, we ensure that all LACs are at least half utilized. Let L be the LAC capacity and D be the total size of all data objects, the number of LACs created from them is always bound in $[D/L, 2D/L]$.

6 Evaluation

We implement our protocols in C++ for evaluation. We rely on the `mcl` library[6] to implement the main cryptographic operations. For 128-bit security, we use the curve BN12-381[7] to realize the PoS scheme in Fig. 8. This gives 48-byte elements from $\mathbb{G}_1$ bytes, 96-byte elements from $\mathbb{G}_2$, and 32-byte elements from $\mathbb{Z}_p$. The same curve is used to implement the BLS signature[8] for protocol OAD^+. We also build the data abstraction layer and plug it on the native Linux file system to understand its performance implication. Since it handles all I/O transparently for the auditing protocols, we evaluate the latter with data entirely in main memory. We use a testbed with Intel E-2174 CPU (3.8GHz, 8 cores) and 64GB RAM. It has a HDD with 4TB capacity and 7200 RPM. All experiments are run with a single core. Unless otherwise specified, each reported data point is averaged over 100 independent runs. For I/O related experiments, we always clear the Linux system page cache to ensure that the OS does not cache files in memory.

TABLE 1: Proof size (in KB) for 1TB data.

Chunk (KB)	2	4	8	16	32	64	128
OAD	0.13						
OAD^+	1.2						
ICPOR	2.1	4.1	8.1	16.1	32.1	64.1	128.1
DPDP	166	164	164	168	180	208	268

6.1 On-chain Costs

We start by analyzing the on-chain efficiency of our protocols. In all experiments below, we set the number of challenged chunks in an audit $c = 128$, a typical security parameter [41]. This provides over 70% and 99% detection probability on each audit, when assuming 1% and 5% data corruption rate, respectively. The detection probability can be made arbitrarily close to 100% by repeated auditing.

Proof size. Our properly instantiated protocols produce small constant-size proofs independent of chunk size. The proof of OAD is simply that of the underlying static construction (Fig. 8), i.e., two elements from $\mathbb{G}_1$ and one from $\mathbb{Z}_p$ that sum up to 128 bytes. The proof of OAD^+ contains additionally c pairs of pseudo indexes and timestamps as well as a short signature, amounting to $128 + 128 * 8 + 48 = 1200$ bytes. To show their advantage, we compare them with two strawman designs, ICPOR and DPDP [21].

ICPOR is a strawman scheme designed by us. It optimizes previous dynamic designs that rely on index translation [32], [51], [53]. They are all based on the seminal construction CPOR [39] but suffer from inefficient update as discussed before. ICPOR is similar but uses succinct linked list (our PIL or that in [9]) to manage index state for data dynamism. DPDP represents the generic ADS-based approach.

Table 1 reports the concrete proof sizes given the same 128-bit security and 1TB file. The proof size of ICPOR equals the size of a chunk plus a tag. The proof size of DPDP approximates the size of a chunk plus c tags and c ADS witnesses each of length $\log(n)$. As can be seen, while ICPOR might be used for on-chain auditing with small chunks, it is unrealistic to use DPDP (and ADS-based designs in general) because its proof is always too large. In contrast to them, OAD and OAD^+ produce much smaller

6. https://github.com/herumi/mcl

7. Pairing-Friendly Curves, https://tools.ietf.org/id/draft-yonezawa-pairing-friendly-curves-02.html

8. https://tools.ietf.org/html/draft-irtf-cfrg-bls-signature-00

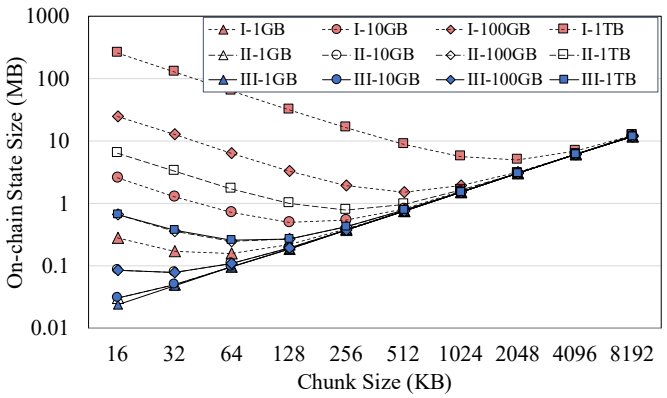


Fig. 10: Overall on-chain state size vs. varying file size and chunk size. In the legends, 'I' = OAD, 'II' = OAD^+ ($m = 100$), 'III' = OAD^+ ($m = 1000$).

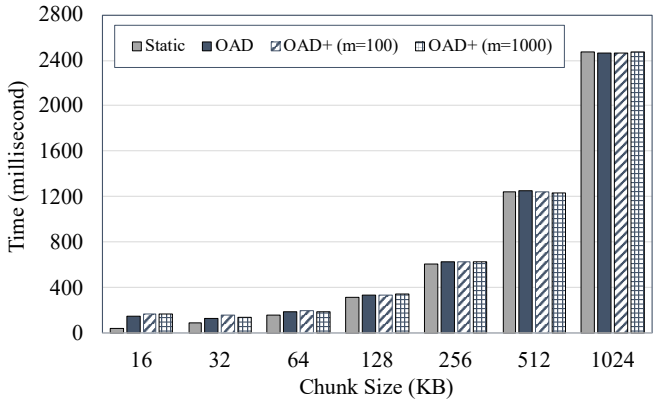


Fig. 11: Proof generation time at the storage node on 1TB data. "Static" indicates the underlying PoS (Fig. 8) used to instantiate our protocols.

proofs that can be well supported by real-world blockchain platforms, e.g., the average block size of Ethereum mostly lie between 30KB–60KB [25].

It goes without saying that the sizable proofs of ICPOR and DPDP already prevent them from being applied to on-chain auditing. Given such fact and space limit, in what follows we will not further discuss other performance metrics of them. We will instead focus on assessing the practicality of our designs from various aspects.

Proof verification cost. We now investigate the feasibility of verifying storage proofs of our protocols on Ethereum. We focus on OAD^+ as its proof is larger and more expensive to verify than OAD. The chunk size has little impact on verification time and we set it as 128KB. Our testbed spends 32ms verifying a proof. Mainstream blockchain platforms such as Etheruem 2.0 and Polkadot use variants of WebAssembly (Wasm) engines for smart contract execution, for Wasm promises near-native code execution performance. Based on recent benchmark results [31], we anticipate an average 50% slowdown when our C++ code is compiled to Wasm. This gives around 48ms verification time on Wasm, which in turn translates to around 183 million cycles (the CPU on our testbed operates in roughly 3.8 billion cycles per second). The Ethereum Wasm gas cost specification suggests that each cycle of a modern commodity CPU equals 0.0045 gas.[9] Consequently, the amount of gas of verifying a proof is estimated to be 0.82 million. Besides, the transaction for submitting the proof will consume roughly 102600 gas (a base cost 21000 gas plus 68 gas per byte [49]). Altogether, the cost of verifying a proof is approximately 0.92 million. This is well within the current 12 million block gas limit[10] and confirms the practicality of our protocols.

As a side note, the above-mentioned local execution of AC's verification function, which takes place at miner nodes, has little impact on the overall blockchain consensus process. In other words, the code execution time of the order of milliseconds is negligible compared to the typical time taken to mine a block, e.g., 10–20 seconds for Ethereum. Thus, we could even have multiple storage proofs verified in a single block, which is impossible for the other two strawman designs.

Update cost. One of the salient features of our protocols is the optimal on-chain data update, which mostly involves updating the index state, i.e., A_{ac} for OAD and IS^- for OAD^+. Recall that the update information $\mathsf{Upd}_{\mathsf{ac}}$ already contains corresponding array indexes, so no linear scan is required to locate the position for update in the index state and hence the cost is trivial. Note that further cost reduction can be obtained by handling updates in batches (see Section 4.2).

On-chain state. Finally, we come to the most important performance aspect of our design. The auditor state of our protocols is comprised of two major parts, the public key of the underlying PoS scheme and the index state. For the chosen parameter, the public key has a size $1.5\times$ the chunk size. OAD^+ has a tunable parameter m to adjust the size of index state IS^-. We consider two realistic values $m = 100$ and $m = 1000$ in the evaluation.

Figure 10 shows the auditor state size for files ranging from 1GB to 1TB. As can be seen, it is possible to find an optimal chunk size that minimizes the state size for a given file size. For 1TB file, OAD^+ produces a 0.78MB state with $m = 100$ for 256KB chunk and a 0.25MB state with $m = 1000$ for 64KB chunk. Remarkably, such small state sizes are even comparable to the proof sizes of ADS-based schemes. This is a dramatic improvement over designs that requires $O(n)$ state. In particular, our optimized OAD already results in a significantly larger state, e.g., 16MB for 256KB chunk. The case for previous constructions is even worse since they are also confined by their proof size. Consider the aforementioned ICPOR protocol and a chunk size of 2KB for it to obtain a proof size comparable to that of OAD^+. ICPOR will then require $(2^{40}/2^{11}) * 8$ bytes $= 4$GB state for 1TB file! The comparison shows a clear edge of our designs. Note that for small to moderate data size, say up to 10GB, OAD with extremely short proof makes it more preferable than OAD^+ for long-term operation.

6.2 Off-chain Costs

We now study costs of the data owner and storage node, paying special attention to the price for data dynamism.

9. https://ewasm.readthedocs.io
10. https://etherscan.io/chart/gaslimit

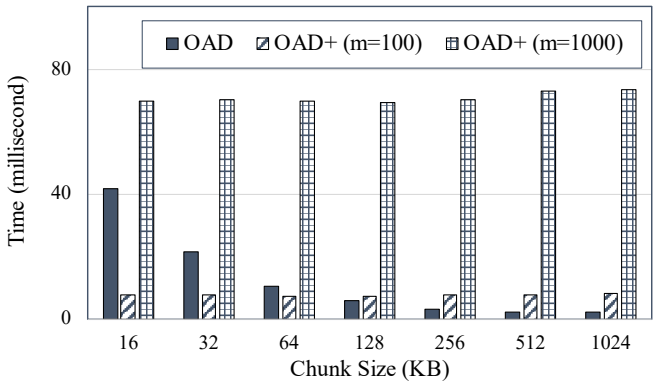


Fig. 12: Update cost at the data owner for inserting one chunk for a 1TB dataset. The position of insertion is $n/2$, representing the average case cost.

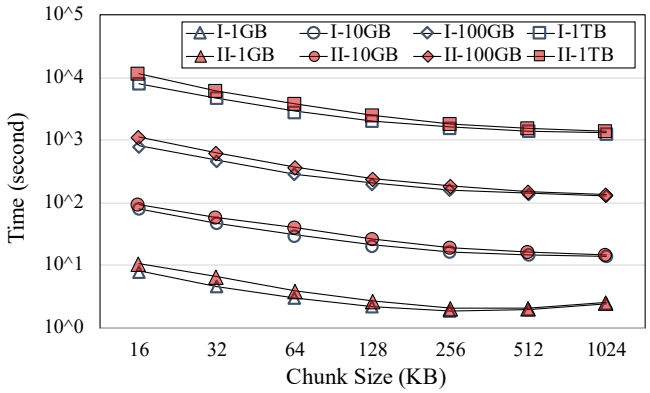


Fig. 13: The setup cost for our auditing protocols at the data owner. 'I' = the underlying PoS (Fig. 8) and 'II' = OAD^+ ($m = 100$, the value of m is insignificant).

Proof generation. Figure 11 reports the proof generation time for 1TB data in one audit. Compared with the static counterpart, the dynamic protocols require linear scan of the state to find correct indexes, and OAD^+ additionally requires the aggregation of signatures over index tuples. As can be seen, the overhead is noticeable for large n resulting from small chunk size. But it quickly diminishes as the chunk size grows. Starting from 64KB chunk, the dynamic protocols perform almost as well as the static one. Previous analysis shows that the minimal on-chain state is obtained by a moderate chunk size between 64KB and 128KB. We thus conclude that with a reasonable setting the overhead of proof generation is marginal.

Data update. Figure 12 reports the data owner's local update cost. Here we consider insertion only since it represents the most expensive one among the three types of update operation. In addition to computing a new tag, the protocols need to traverse the PIL of corresponding index state to find the right position for insertion. The evaluation results suggest that the traversal dominates the update cost of OAD but not OAD^+. This is explained by the latter's two-layer structure that reduces the search time by a factor of m. The main cost of OAD^+ comes from re-signing all index tuples for the affected segment. Since the tuples are small, i.e., 20 bytes each (four 4-byte integers p_i, t_i, ρ_k, τ_k and two 2-byte integers l, η_k), signing them is inexpensive, e.g., $<$ 80ms for 1000 signatures. The data owner needs to pass all new signatures to the storage node. Given 48-byte signatures and a preferred chunk size 64KB, the bandwidth inflation is 75% for $m = 1000$. This should be acceptable to the communication between the off-chain parties.

How to improve off-chain update efficiency is an interesting research direction. For example, it is possible to replace the aggregate signature scheme with a trapdoorless RSA accumulator [12] to authenticate index tuples. This can provide asymptotically better update performance: $O(1)$ for OW-side computation, $O(m)$ for SN-side computation, and $O(1)$ communication between them. However, the resulting on-chain costs will increase drastically. Since n/m accumulator values (one for a segment) should be stored by the auditor, a large constant factor will be added to the on-chain state size, e.g., 384 bytes (for 128-bit security) compared with the 10 bytes of IS^-. Also, each update requires writing an updated large accumulator value to the blockchain. We leave the study of an enhanced solution as a future work.

One-time setup. Figure 13 reports the setup time at the data owner. The cost of OAD should be the same as the static construction because its index state initialization is trivial. OAD^+ incurs extra cost to sign index tuples, but its overall setup cost is still dominated by the underlying PoS.

6.3 Overhead of Data Abstraction

Table 2 summarizes the datasets used in our experiments. Four of them are generated with FileBench [44] following a gamma distribution with mean file size 1MB, 2MB, 5MB, and 10MB. The last one is the Linux code repository cloned from GitHub with the `.git` folder stripped. We evaluate the overheads of the data abstraction layer throughout different the life cycle of a dynamic auditing protocol. In what follows the term chunk refer to LAC.

Setup phase. The data owner needs to construct the data abstraction layer locally during setup. We evaluate the construction time for different chunk sizes (Fig. 14). The results show that even for the smallest chunk size that results in most chunks, it takes just 10 seconds to set up the 1TB dataset. This low cost is due to the fact that constructing the abstraction layer only requires reading the files' metadata but not their actual contents. We also measure the total size of chunks in the unit of triplets. At the 128KB chunk size, there are roughly 8 millions and 100 thousands triplets for the LB-1TB and Linux dataset, respectively. Given 40-byte triplet, the numbers translate to 320MB and 4MB, or 0.032% and 0.1% of the datasets. Thus, the data abstraction layer is practically small enough to be kept in main memory.

Auditing phase. To generate proofs, the storage node must load the actual data pointed by the chunks into memory from hard disk, by invoking the underlying storage system. This is where the main I/O cost for data auditing comes from. We report the time to load random chunks in Fig. 15. In theory, the I/O cost should be proportional to the chunk size, and this general trend is visible in the figure. Yet, results on the two larger datasets show an opposite direction for smaller chunks. This is due to the sequential scan to access the chunks. We can improve the performance by storing pointers to the chunks in a data structure that supports

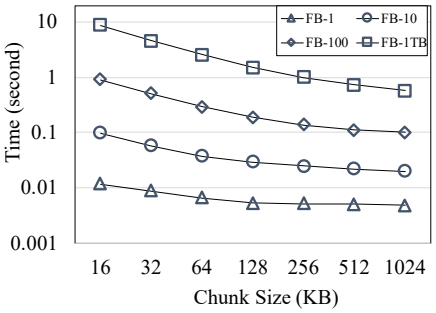


Fig. 14: Construction time for the data abstraction layer with varying chunk sizes over four datasets.

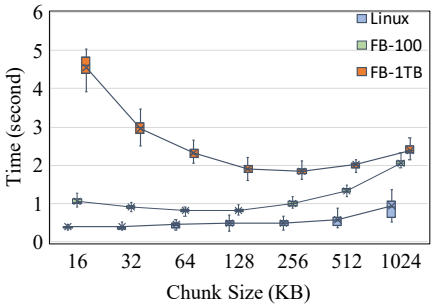


Fig. 15: Time to load $c = 128$ challenged chunks into memory for proof generation in an audit.

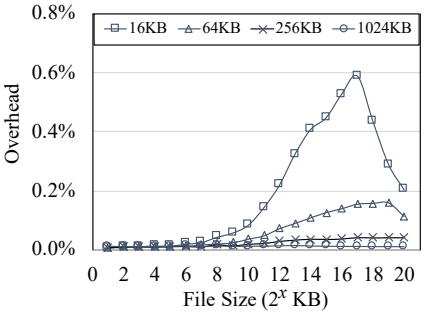


Fig. 16: Worst-case overhead of random file writes on the FB-1TB dataset at the storage node, with four chunk sizes.

TABLE 2: Summary of file system datasets.

Dataset	FB-1	FB-10	FB-100	FB-1TB	Linux
Size	1GB	10GB	100GB	1TB	1.02GB
Num. files	968	5531	21426	103356	66458
Num. dirs	27	63	270	591	4,390
Dir. depth	2	4	7	10	9

fast lookup. Overall, even the current loading time on the order of several seconds does not constitute a practical performance issue, as for on-chain auditing we should set a lenient timeout on the order of minutes for a honest storage node's response message to be recorded on the blockchain.

Update phase. We also measure the overhead of writing updated file at the storage node, which entails updating relevant triplets in chunks. The estimated overhead is relative to the time to simply write the files to hard disk. We consider file sizes ranging from 1KB to 1GB and for each size we randomly sample files from the FB-1TB dataset. The results in Fig. 16 indicate that the overhead is negligible ($\leq 1\%$) for all file sizes and chunk sizes.

7 Conclusion

The emerging decentralized storage paradigm has shown great potentials. To motivate wary users, such a system must provide strong data integrity assurance while supporting full data dynamism for usability, which has rarely been studied before. This paper offers a thorough investigation into the inadequacies of existing data auditing solutions in fulfilling the new paradigm. To fill the void, we develop a family of on-chain auditing protocols optimized for dynamic data and demonstrate their practical efficiency. As the momentum of decentralized storage with mandatory auditability continues, we hope that our results can spur more research on the usability aspects of this exciting technology.

Acknowledgments

This work was supported in part by the National Key R&D Program of China (2020YFB1005500), partially supported by the Research Grants Council of Hong Kong under Grant CityU 11218521 and R6021-20F, and the Research Fellowship Scheme 2021/22 (Prof. Cong Wang), by Shenzhen Municipality Science and Technology Innovation Commission (grant no. SGDX20201103093004019, CityU), and also by the National Natural Science Foundation of China under Grants 61572412, U20B2049, and 61822207.

References

[1] "Orbitdb: Peer-to-peer databases for the decentralized web," Online at: https://orbitdb.org.
[2] "Filecoin: A decentralized storage network," *white paper*, 2017.
[3] A. Aghayev, S. Weil, M. Kuchnik, M. Nelson, G. R. Ganger, and G. Amvrosiadis, "File systems unfit as distributed storage backends: lessons from 10 years of ceph evolution," in *Proc. of ACM SOSP*, 2019, pp. 353–369.
[4] G. Anthoine, J. Dumas, M. Hanling, M. de Jonghe, A. Maignan, C. Pernet, and D. S. Roche, "Dynamic proofs of retrievability with low server storage," in *Proc. of USENIX Security*, 2021.
[5] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proc. of ACM CCS*, 2007.
[6] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. of ACM SecureComm*, 2008.
[7] G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," in *Proc. of ASIACRYPT*, 2009.
[8] T. Bana and J. Olio, "So you're a storage node operator. which satellites do you trust?" Online at: https://storj.io/blog/2019/08/so-youre-a-storage-node-operator.-which-satellites-do-you-trust/, 2019.
[9] A. Barsoum and A. Hasan, "Enabling dynamic data and indirect mutual trust for cloud computing storage systems," *IEEE transactions on parallel and distributed systems*, vol. 24, no. 12, pp. 2375–2385, 2012.
[10] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *Proc. of USENIX Security*, 2014.
[11] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 2012.
[12] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to iops and stateless blockchains," in *Proc. of CRYPTO*, 2019.
[13] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proc. of EUROCRYPT*, 2003.
[14] M. Campanelli, D. Fiore, N. Greco, D. Kolonelos, and L. Nizzardo, "Incrementally aggregatable vector commitments and applications to verifiable decentralized storage," in *Proc. of ASIACRYPT*, S. Moriai and H. Wang, Eds., 2020.
[15] D. Cash, A. Küpçü, and D. Wichs, "Dynamic proofs of retrievability via oblivious ram," in *Proc. of Eurocrypt*, 2013, pp. 279–295.
[16] E. Cecchetti, B. Fisch, I. Miers, and A. Juels, "Pies: Public incompressible encodings for decentralized storage," in *Proc. of ACM CCS*, 2019.
[17] L. Cheng, Y. Hu, and P. P. C. Lee, "Coupling decentralized key-value stores with erasure coding," in *Proc. of ACM SoCC*, 2019.

[18] L. C. David Vorick, "Sia: Simple decentralized storage," *white paper*, 2014.

[19] Y. Du, H. Duan, A. Zhou, C. Wang, M. Au, and Q. Wang, "Enabling secure and efficient decentralized storage auditing with blockchain," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 1–1, 2021.

[20] Y. Du, H. Duan, A. Zhou, C. Wang, M. H. Au, and Q. Wang, "Towards privacy-assured and lightweight on-chain auditing of decentralized storage," in *Proc. of IEEE ICDCS*, 2020.

[21] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 4, p. 15, 2015.

[22] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proc. of ACM CCS*, 2009.

[23] E. Esiner, A. Kachkeev, S. Braunfeld, A. Küpçü, and Ö. Özkasap, "Flexdpdp: Flexlist-based optimized dynamic provable data possession," *ACM Transactions on Storage (TOS)*, vol. 12, no. 4, p. 23, 2016.

[24] M. Etemad and A. Küpçü, "Generic dynamic data outsourcing framework for integrity verification," *ACM Comput. Surv.*, vol. 53, no. 1, 2020.

[25] Etherscan. (2021) Ethereum average block size chart. [Online]. Available: https://etherscan.io/chart/blocksize/

[26] Filecoin, "Filecoin research," Online at: https://github.com/filecoin-project/research, 2020.

[27] D. Francati, G. Ateniese, A. Faye, A. M. Milazzo, A. M. Perillo, L. Schiatti, and G. Giordano, "Audita: A blockchain-based auditing framework for off-chain storage," *arXiv preprint arXiv:1911.08515*, 2019.

[28] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *J. ACM*, vol. 43, no. 3, p. 431–473, 1996.

[29] W. Guo, S. Qin, F. Gao, H. Zhang, W. Li, Z. Jin, and Q. Wen, "Dynamic proof of data possession and replication with tree sharing and batch verification in the cloud," *IEEE Transactions on Services Computing*, vol. 1, no. 1, pp. 1–1, 2020.

[30] Z. Hao, S. Zhong, and N. Yu, "A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability," *IEEE transactions on Knowledge and Data Engineering*, vol. 23, no. 9, pp. 1432–1437, 2011.

[31] A. Jangda, B. Powers, E. D. Berger, and A. Guha, "Not so fast: analyzing the performance of webassembly vs. native code," in *Proc. of USENIX ATC*, 2019.

[32] H. Jin, H. Jiang, and K. Zhou, "Dynamic and public auditing with fair arbitration for cloud data," *IEEE Transactions on Cloud Computing*, vol. 6, no. 3, pp. 680–693, 2016.

[33] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Proc. of ASIACRYPT*, 2010.

[34] S. Keelveedhi, M. Bellare, and T. Ristenpart, "Dupless: server-aided encryption for deduplicated storage," in *Proc. of USENIX Security*, 2013.

[35] H. Kopp, C. Bösch, and F. Kargl, "Koppercoin–a distributed file storage with financial incentives," in *Proc. of International Conference on Information Security Practice and Experience*. Springer, 2016, pp. 79–93.

[36] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proc. of CRYPTO*, 1988.

[37] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Authenticated hash tables based on cryptographic accumulators," *Algorithmica*, vol. 74, no. 2, pp. 664–712, 2016.

[38] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. H. Nyang, and D. Mohaisen, "Exploring the attack surface of blockchain: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 1, no. 1, pp. 1–1, 2020.

[39] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. of ASIACRYPT*, 2008.

[40] J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo, "An efficient public auditing protocol with novel dynamic structure for cloud data," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2402–2415, 2017.

[41] E. Shi, E. Stefanov, and C. Papamanthou, "Practical dynamic proofs of retrievability," in *Proc. of ACM CCS*, 2013.

[42] E. Stefanov, M. van Dijk, A. Juels, and A. Oprea, "Iris: A scalable cloud file system with efficient integrity checks," in *Proc. of ACM ACSAC*, 2012, pp. 229–238.

[43] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *Proc. of IEEE S&P*, 2017.

[44] V. Tarasov, E. Zadok, and S. Shepler, "Filebench: A flexible framework for file system benchmarking," *Usenix ;login:*, vol. 41, no. 1, 2016.

[45] H. Tian, Y. Chen, C.-C. Chang, H. Jiang, Y. Huang, Y. Chen, and J. Liu, "Dynamic-hash-table based public auditing for secure cloud storage," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 701–714, 2015.

[46] C. Wang, S. S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362–375, 2011.

[47] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward secure and dependable storage services in cloud computing," *IEEE transactions on Services Computing*, vol. 5, no. 2, pp. 220–232, 2011.

[48] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 847–859, 2011.

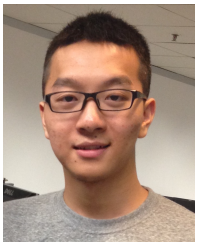
[49] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2014.

[50] A. Yang, J. Xu, J. Weng, J. Zhou, and D. S. Wong, "Lightweight and privacy-preserving delegatable proofs of storage with data dynamics in cloud storage," *IEEE Transactions on Cloud Computing*, 2018.

[51] K. Yang and X. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1717–1726, 2012.

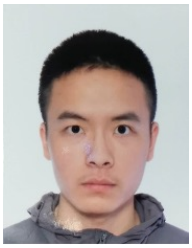
[52] J. Yuan and S. Yu, "Proofs of retrievability with public verifiability and constant communication cost in cloud," in *Proc. of ACM CloudComputing*, 2013.

[53] Y. Zhu, G.-J. Ahn, H. Hu, S. S. Yau, H. G. An, and C.-J. Hu, "Dynamic audit services for outsourced storages in clouds," *IEEE Transactions on Services Computing*, vol. 6, no. 2, pp. 227–238, 2011.



Huayi Duan received his B.S. degree (2015) and Ph.D. degree (2020), both from City University of Hong Kong. His research interests span network security, data integrity and privacy, trustworthy and confidential computing, and decentralized technologies. He is a member of IEEE.

Yuefeng Du received the BSc degree in computer science from City University of Hong Kong, in 2018. He is working toward the PhD degree at the City University of Hong Kong. His research interests include applied cryptography, blockchain and network security. He is a student member of the IEEE.



Leqian Zheng received the BSc degree in computer science from City University of Hong Kong, in 2021. He is working toward the PhD degree at the City University of Hong Kong. His research interests include blockchain, optimization, and machine learning security.

Cong Wang is currently a Professor with the Department of Computer Science, City University of Hong Kong. He received his Ph.D. degree in the Electrical and Computer Engineering from Illinois Institute of Technology, USA, M.E. degree in Communication and Information System, and B.E. in Electronic Information Engineering, both from Wuhan University, China. His current research interests include data and computation outsourcing security in the context of cloud computing, blockchain and decentralized application, network security in emerging Internet architecture, multimedia security, and privacy-enhancing technologies in the context of big data and IoT. He is one of the Founding Members of the Young Academy of Sciences of Hong Kong. He is a co-recipient of the Best Paper Award of IEEE ICDCS 2020, the Best Student Paper Award of IEEE ICDCS 2017, the Best Paper Award of IEEE ICPADS 2018, MSN 2015 and CHINACOM 2009. His research has been supported by multiple government research fund agencies, including National Natural Science Foundation of China, Hong Kong Research Grants Council, and Hong Kong Innovation and Technology Commission. He serves/has served as associate editor for IEEE Transactions on Dependable and Secure Computing (TDSC), IEEE Internet of Things Journal (IoT-J) and IEEE Networking Letters, and TPC co-chairs for a number of IEEE conferences/workshops. He is a Fellow of the IEEE and a member of the ACM.

Man Ho Au received the B.Eng. and M.Phil. degrees from the Department of Information Engineering, The Chinese University of Hong Kong, in 2003 and 2005, respectively, and the Ph.D. degree from the School of Computer Science and Software Engineering, University of Wollongong, in 2009. He is currently an Associate Professor with the Department of Computer Science, The University of Hong Kong (HKU). Prior to joining HKU, he was an Associate Professor with the Department of Computing, The Hong Kong Polytechnic University. He has published over 160 refereed articles in top journals and conferences, including CRYPTO, ASIACRYPT, ACM CCS, ACM SIGMOD, NDSS, IEEE TIFS, TC, and TKDE. His research interests include applied cryptography, information security, blockchain technology, and related industrial applications. He was a recipient of the 2009 PET Runner-Up Award for outstanding research in privacy enhancing technologies. His digital signature technology has been adopted by the Hyperledger Fabric Project. He is also an Expert Member of the China delegation of ISO/IEC JTC 1/SC 27 Working Group 2: Cryptography and Security Mechanisms and a Committee Member of the Research and Development Division, Hong Kong Blockchain Society.

Qian Wang received the Ph.D. degree from the Illinois Institute of Technology, USA. He is currently a Professor with the School of Cyber Science and Engineering, Wuhan University. His research interests include AI security, data storage, search and computation outsourcing security and privacy, wireless systems security, big data security and privacy, and applied cryptography. He is a Member of ACM. He received the National Science Fund for Excellent Young Scholars of China in 2018. He is also an expert under National “1000 Young Talents Program” of China. He was a recipient of the 2018 IEEE TCSC Award for Excellence in Scalable Computing (Early Career Researcher), and the 2016 IEEE Asia–Pacific Outstanding Young Researcher Award. He is also a co-recipient of several best paper and best student paper awards from the IEEE ICDCS’17, the IEEE TrustCom’16, WAIM’14, and the IEEE ICNP’11. He serves as an Associate Editors for the IEEE Transactions on Dependable and Secure Computing (TDSC) and the IEEE Transactions On Information Forensics and Security (TIFS).