Efficient Dynamic Proof of Retrievability for Cold Storage

Tung Le
Virginia Tech
tungle@vt.edu

Pengzhi Huang
Cornell University
ph448@cornell.edu

Attila A. Yavuz
University of South Florida
attilaayavuz@usf.edu

Elaine Shi
CMU
runting@gmail.com

Thang Hoang
Virginia Tech
thanghoang@vt.edu

***Abstract*—Storage-as-a-service (STaaS) permits the client to outsource her data to the cloud, thereby reducing data management and maintenance costs. However, STaaS also brings significant data integrity and soundness concerns since the storage provider might not keep the client data intact and retrievable all the time (e.g., cost saving via deletions). Proof of Retrievability (PoR) can validate the integrity and retrievability of remote data effectively. This technique can be useful for regular audits to monitor data compromises, as well as to comply with standard data regulations. In particular, cold storage applications (e.g., MS Azure, Amazon Glacier) require regular and frequent audits with less frequent data modification. Yet, despite their merits, existing PoR techniques generally focus on other metrics (e.g., low storage, fast update, metadata privacy) but not audit efficiency (e.g., low audit time, small proof size). Hence, there is a need to develop new PoR techniques that achieve efficient data audit while preserving update and retrieval performance.**

In this paper, we propose Porla, a new PoR framework that permits efficient data audit, update, and retrieval functionalities simultaneously. Porla permits data audit in both private and public settings, each of which features asymptotically (and concretely) smaller audit-proof size and lower audit time than all the prior works while retaining the same asymptotic data update overhead. Porla achieves all these properties by composing erasure codes with verifiable computation techniques which, to our knowledge, is a new approach to PoR design. We address several challenges that arise in such a composition by creating a new homomorphic authenticated commitment scheme, which can be of independent interest. We fully implemented Porla and evaluated its performance on commodity cloud (i.e., Amazon EC2) under various settings. Experimental results demonstrated that Porla achieves *two* to *four* orders of magnitude smaller audit proof size with $4\times$–$18000\times$ lower audit time than all prior schemes in both private and public audit settings at the cost of only $2\times$–$3\times$ slower update.

I. Introduction

Storage-as-a-Service (STaaS) provides a sophisticated data storage facility and infrastructure for clients to outsource their data to the cloud, thereby reducing expensive data management, maintenance, and archival costs [30]. Despite its merits, STaaS has posed significant concerns regarding the data soundness to the client. When a client outsources her data to a remote storage provider, it is not clear whether all data will be kept intact and retrievable over time. Data loss can happen due to unwanted accidents (e.g., hardware failure) or adversarial behaviors. For instance, the adversary may strategically omit some important data update, or delete certain parts of the data that are rarely accessed to reduce storage overhead and monetary maintenance costs.

Under a standard Service Level Agreement (SLA), a reliable and trustworthy storage provider is expected to comply with standard data regulations (e.g., [35], [45], [39]) by performing a regular audit to ensure persistent data integrity and freshness. Audit-proof archiving [2] is one of the best practices for secure digital data storage that generally maintains sensitive information (e.g., personal, health). A fine-grained audit on a regular basis is necessary to continuously monitor the system activities against potential threats [33]. In particular, cold storage and archival applications (e.g., Amazon Glacier [1], MS Azure Archive [3], or Blob [13]) maintain a large amount of archival data that is for long-term maintenance [14], [10]. In such applications, the archives are rarely updated but must be periodically audited to ensure their availability and trustworthiness. Both the size of the data and the number of audit logs grow significantly over time. Hence, it is critical that the size of cryptographic audit tags is small to avoid storage bottlenecks in cold storage and digital archival systems.

Several cryptographic techniques have been developed [11], [31] to permit effective data integrity audit. Provable Data Possession (PDP) [11] allows a client to check whether her data is kept intact by the storage server. Despite its merits, PDP only ensures the integrity of *most* of the data, but not all. By deleting a small portion of the data, the adversarial server can still bypass the audit with a high probability. Proof of Retrievability (PoR) [31] achieves a stronger security notion in the sense that the audit can provably tell whether *all* the data is intact and retrievable or not. While preliminary PoR schemes are static (e.g., [40], [17], [31]), recent Dynamic PoR (DPoR) constructions (e.g., [7], [15], [20], [41], [43]) have enabled data updatability and auditability simultaneously. Most proposed DPoR schemes rely on coding theory and cryptographic techniques as the main building blocks such as error correction code (ECC) [20], [41], [43], Oblivious RAM (ORAM) [20], and verifiable computation [7], [15].

Each of the proposed DPoR schemes to date features unique properties with special characteristics (e.g., strong privacy, update efficiency, low storage). However, most of them are not ideal for audit-intensive use cases. Specifically, apart from the auditability, ORAM-based DPoR [20] can also hide data access patterns (e.g., read/write); however, it incurs significantly high communication overhead during an audit. Shi et al. proposed a DPoR scheme based on a locally

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.23307
www.ndss-symposium.org

TABLE I: Comparison of our Porla framework over state-of-the-art.

Scheme	Write			Audit			Storage	
	Bandwidth	Client cost	Server cost	Proof size	Client cost	Server cost	Server	Client
SSP13 [41]	$\beta + \lvert\pi_{\mathsf{vc}}\rvert + \mathcal{O}(\lambda \log N)$	$1\mathcal{U}_{\mathsf{vc}} + \mathcal{O}(\lambda\beta)\mathbb{F} + \mathcal{O}(\log N)\mathsf{E}$	$1\mathcal{U}_{\mathsf{vc}} + \mathcal{O}(\beta \log N)\mathbb{F}$	$\mathcal{O}(\beta + \lambda^2 \log N)$	$\mathcal{O}(\beta\lambda)\mathbb{F} + \mathcal{O}(\lambda \log N)\mathsf{E}$	$\mathcal{O}(\beta\lambda \log N)\mathbb{F}$	$\mathcal{O}(N\beta)$	$\mathcal{O}(\beta + \lambda)$
CKW17 [20]	$\mathcal{O}(\beta\lambda^2 \log^2 N)$	$\mathcal{O}(\beta\lambda^2 \log^2 N)\mathsf{E}$	$\mathcal{O}(\beta\lambda \log^2 N)$	$\mathcal{O}(\beta\lambda^2 \log^2 N)$	$\mathcal{O}(\beta\lambda^2 \log^2 N)\mathsf{E}$	$\mathcal{O}(\beta\lambda^2 \log^2 N)$	$\mathcal{O}(N\beta)$	$\mathcal{O}(\lambda)$
ADJ+21 [7]	$\beta + \lvert\pi_{\mathsf{vc}}\rvert$	$1\mathcal{U}_{\mathsf{vc}}$	$1\mathcal{U}_{\mathsf{vc}}$	$\mathcal{O}(\lambda\sqrt{\beta N})$	$\mathcal{O}(\lambda\sqrt{\beta N})\mathbb{F}$	$\mathcal{O}(N\beta)\mathbb{F}$	$\mathcal{O}(N\beta)$	$\mathcal{O}(\sqrt{\beta N})$
Our $\mathsf{Porla}_{\mathsf{kzg}}$	$\beta + \lvert\pi_{\mathsf{vc}}\rvert$	$1\mathcal{U}_{\mathsf{vc}} + \mathcal{O}(\beta)\mathbb{G}$	$1\mathcal{U}_{\mathsf{vc}} + \mathcal{O}(\beta \log N)\mathbb{F} + \mathcal{O}(\log N)\mathbb{G}$	$3\lvert\mathbb{G}\rvert$	$\mathcal{O}(\lambda \log N)\mathsf{PRF} + 1\mathsf{e}$	$\mathcal{O}(\beta\lambda \log N)\mathbb{F} + \mathcal{O}(\lambda \log N + \beta)\mathbb{G}$	$\mathcal{O}(N\beta)$	$\mathcal{O}(\lambda)$
Our $\mathsf{Porla}_{\mathsf{ipa}}$				$2\log_2 \beta\lvert\mathbb{G}\rvert + 2\lvert\mathbb{F}\rvert$	$\mathcal{O}(\lambda \log N)\mathsf{PRF} + \mathcal{O}(\beta)\mathbb{G}$			

This table presents the cost of private audit. For public audit setting, we refer to Table II. N = # data blocks, β = block size, λ = security parameter. Client/server cost includes client/server computation and disk I/O access. $\mathbb{F}$ means finite field arithmetic operations; $\mathbb{G}$ means group operations, E means symmetric operations (i.e., encryption/decryption); PRF means PRF operations; e means pairing operations. $\lvert\pi_{\mathsf{vc}}\rvert$ is the membership proof size and $\mathcal{U}_{\mathsf{vc}}$ is the cost to update the vector commitment with a new element of the underlying VC scheme, respectively. See §V-D for detailed analysis of our schemes.

updatable ECC that departs from ORAM to reduce the audit bandwidth cost [41]. However, its audit proof size grows linearly to the block size and logarithmically to the data size, and is significantly large in the public audit setting. This may not be desirable for applications that require auditing on large databases with large block sizes (e.g., image/video storage) and maintaining the audit logs for digital forensics. Finally, verifiable computation-based techniques [7], [15] offer efficient data updates with small server storage. However, their audit cost is expensive, since the entire data needs to be processed. This may not be ideal for situations where periodic audits over large-scale databases are needed.

Given that there is a lack of a DPoR design focusing on optimizing the audit overhead, we ask the following question:

Can we design a new DPoR scheme that minimizes the audit cost (i.e., proof size, end-to-end latency) for audit-critical applications such as cold-storage, while maintaining a reasonable data update performance over state-of-the-art?

A. Our Results and Contributions

We answer this question with a new DPoR framework called Porla that achieves low audit processing with minimal audit log size while retaining asymptotically efficient data update. Our construction makes use of both coding theory and verifiable computation techniques which, to our knowledge, is the first that considers such an integrated approach in DPoR design. We introduce two instantiations from our generic scheme via succinct polynomial commitment schemes (e.g., KZG [32]) and via an Inner-Product Argument (IPA) [18].

- **Minimal audit proof size.** Porla offers a minimal audit proof size *regardless* of the database size. The proof size of our KZG-based Porla only costs three group elements, while that of the IPA-based Porla only grows logarithmically with the data block size plus two field elements. This results in *two* to *four* orders of magnitude smaller proof size than all prior DPoRs (§VII-C). To our knowledge, Porla is the first to achieve a constant audit proof size.
- **Low audit time.** Our scheme causes a sub-linear audit overhead at both client and server, thereby achieving much lower processing latency than verifiable computation-based techniques (e.g., [7]). Experimental results on a real cloud platform demonstrated that our scheme achieves $4\times$–$18000\times$ faster audit time than all prior approaches (§VII-C).
- **Low client storage.** Our scheme only requires the client to store a λ-bit master key and a counter (where λ is the security parameter). This is much more efficient than [7], where the client storage cost is proportional to the *square-root* of the database size.
- **Efficient public audit.** Our scheme can enable public audit with a small extra overhead compared with prior schemes (e.g., [41], [7]). Specifically, it incurs an extra of $\mathcal{O}(\lambda \log N)$ group elements to the audit proof size and $\mathcal{O}(\lambda \log N)$ group operations to the client processing (where λ is a security parameter), compared with $\mathcal{O}(\lambda \log N)$ data blocks in [41] and $\mathcal{O}(\sqrt{N\beta})$ group operations in [7], where N is the number of data blocks and β is the block size (see Table II and §VII-F). Concretely, our public audit proof size is one to two orders of magnitude smaller and our audit delay is $45\times$–$11700\times$ faster than all prior works (e.g., [41], [7]) (see §VII-F), where it only costs around 1 second to audit a 2 TB database. Due to the small proof size, Porla can be a potential candidate to develop other useful applications such as Proof of Space [24], [38] as an alternative consensus mechanism, or digital fair exchange protocols [5], [19].
- **Techniques: "homomorphic authenticated commitment".** To construct our DPoR scheme with low audit overhead, we come up with a new authentication technique that can be of independent interest and can lead to other interesting applications. We construct a homomorphic Message Authentication Code (MAC) for discrete log-based (homomorphic) commitment. Our MAC offers homomorphic property in the sense that the authentication tags can be aggregated into a single tag according to some linear combination. We prove that our MAC achieves the existential unforgeability under chosen message attack (§IV).
- **Efficient data update.** We inherit all the asymptotic overhead of data update in [41]. In fact, the client's overhead and bandwidth in our scheme are concretely lower than [41] during the online update thanks to the pre-computation properties of our MAC scheme, which permits the MAC update token to be pre-computed and transmitted ahead of time in the offline phase (see Remark 1).

Table I compares our DPoR with state-of-the-art schemes. We analyzed the security of our proposed techniques and formally proved that they satisfy the standard security notions (i.e., unforgeability, authenticity, retrievability). We fully implemented our schemes and evaluated their performance on commodity Amazon EC2 clouds. Experimental results showed that our technique outperforms prior works in all audit metrics (e.g., proof size, delay, public audit setting) (see §VII). Our implementation will be open-sourced for wide adaptation.

B. Technical Highlights

Our scheme relies on the blueprint design of a locally updatable code called incrementally constructible code (ICC) proposed in [41]. We start by presenting its high-level idea.

A brief overview of ICC code. All DPoR schemes with sublinear audit cost (e.g., [41], [20]) that harness ECC to encode the data is based on the elegant observation by Juels et al. [31]: If only a small portion of the ECC codeword is damaged, ECC can help to recover the original data. Otherwise, if it exceeds the correction threshold, by checking the authenticity of $\mathcal{O}(\lambda)$ random codeword blocks, one of them will likely be the corrupted block that cannot bypass the authenticity check and therefore will fail the audit. Despite its usefulness in audit, ECC poses difficulty in the update because a small change in the original data will result in rebuilding the entire codeword. To address this issue, ICC code was proposed, which comprises multiple ECC codewords with different sizes to support data updates over time, in which smaller ECC codewords are rebuilt more frequently than large codewords which are rebuilt only after a certain number of updates. By taking amortization into account, the (amortized) cost per data update will become small (i.e., logarithmic to the total data size). To perform an audit on ICC, the client samples $\mathcal{O}(\lambda)$ random positions per ECC codeword and verifies their authenticity as in any ECC-based DPoR. Shi et al. [41] showed that the challenged codeword blocks can be aggregated into a single block using a random linear combination to reduce bandwidth overhead. However, it still requires transmitting the aggregated block itself and the authentication tags of individual blocks being aggregated for authenticity check. This may incur a large audit proof size and bandwidth overhead given that the block size and the database are large.

Idea 1: Transmit the block commitment, and prove the knowledge of opening of commitment. Our first idea is that instead of transmitting the aggregated block, the server can commit to it using a succinct commitment scheme (e.g., polynomial commitment), and then prove the knowledge of the opening of the commitment. This permits us to reduce the audit proof size to the size of proving the opening of commitment, which can be as small as just a few group elements. However, there are some challenges to realizing this idea. Specifically, how to verify if the codeword block the server commits to is indeed the aggregation of the original blocks being challenged? Although this can be done by applying the idea in [41] by creating an authentication tag for the commitment of each block, it still requires transmitting the authentication tags of individual commitments, thereby increasing the audit-proof size and bandwidth. Can we do it better?

Idea 2: Create homomorphic authenticated commitment to further reduce audit proof size. We develop a new MAC scheme for polynomial commitment, which permits to verify the authenticity of the commitment of the aggregated block without attaching multiple authentication tags to the audit-proof. This is achieved by making the authenticated commitment become homomorphic in the sense that not only the commitment of individual codeword blocks can be linearly combined but also their authentication tags can be aggregated accordingly. This permits us to verify the authenticity of the commitment of the aggregated block in Idea 1 with just a single authentication tag, given that the aggregation is based on a random linear combination. Note that several homomorphic MACs were proposed in the literature (e.g., [6], [21], [41]). However, they were designed for different message structures (e.g., network coding, circuits). In our setting, the message to be authenticated is a commitment and, therefore, it is not suitable to directly use these techniques. Thus, we design a new homomorphic MAC scheme for the commitment by exploiting its algebraic structures (e.g., group elements).

Putting everything together. By combining two ideas, we can see that the audit proof now only contains a commitment, a proof of opening, and an authentication tag, and thus its size is minimal. We present the high-level workflow of our audit as follows. During the setup phase, the client encodes the database with ICC code and creates the authenticated commitment for each codeword block. During the audit, the client first samples $\mathcal{O}(\lambda)$ random challenged blocks per ECC codeword. The server then aggregates all the challenged blocks to a single block according to a random linear combination indicated by the client, then commits to the aggregated block and performs the same linear combination over the corresponding authentication tags of the commitments of the challenged blocks. The server sends the aggregated commitment and the aggregated tag to the client, who in turn verifies the authenticity of the commitment against its MAC tag. Finally, the client attests to the server's knowledge of the opening of the aggregated commitment, which, thanks to the random linear combination, can only be bypassed if the server maintains the knowledge of individual challenged codeword blocks.

At the high-level idea, we can see that our DPoR designs make use of both ECC and verifiable computation techniques (i.e., verifiable polynomial delegation). It is worth noting that we are the first to consider such a combination, which has never been explored in the literature. Specifically, all prior DPoR schemes rely solely on either ECC [20], [41] or verifiable computation [7] (but not both), which results in either large audit proof size or linear computation cost, respectively. We overcome these limitations with a new DPoR design that inherits desirable properties of both ECC and verifiable computation techniques to achieve a highly efficient audit (i.e., small proof size, sublinear computation) while maintaining a reasonable data update overhead. We solve several technical challenges that arise when bridging both ECC and VC techniques together by creating a novel efficient authentication mechanism for homomorphic commitments.

C. Related Work

PDP and PoR are highly related to each other, both permit the client to attest that whether her outsourced data is kept intact by the server without retrieving it. The main difference between PDP and PoR is that PDP only ensures the integrity of *most* data, while PoR ensures *all* data achieves integrity. Both PoR and PDP were first suggested at almost the same time in two independent works by Ateniese et al. [11] and Juels et al. [31]. We review PoR/PDP schemes that are the most relevant to our constructions with unique properties.

Static data. Early PDP/PoR constructions permit integrity verification over static data (i.e., no update) [31], [11], [23], [49], [34]. Most of these schemes focus on improving communication complexity and achieving precise security.

Dynamic data. Several constructions attempt to enable data updatability while retaining integrity auditability efficiently. It

is challenging to permit both data updatability and auditability in PDP and PoR schemes. For example, Dynamic PDP schemes in [25], [47], [12] permit updates, but their audit protocol does not achieve the same security notion as PoR. On the other hand, since most audit-efficient PoR schemes (e.g., [31]) encode original data with ECC, updating a small piece of data requires rebuilding the entire codeword, which is costly. Therefore, Cash et al. [20] suggested encoding original data with multiple small codewords and using ORAM to obliviously update the codeword. Shi et al. in [41] proposed a locally updatable ECC that permits an update over a data block to rebuild only some small codewords. There are some constructions (e.g., [15], [7]) that depart from ECC and rely solely on verifiable computation techniques; however, they require processing the entire database for integrity check.

Public audit. Some constructions permit public audit, in which the data integrity can be verified by a public auditor without the data owner's intervention. Some constructions [9], [8] are designed for public audit only, while there are some works that offer both private and public audit capabilities (e.g., [41], [7]) (including our work).

Trusted proxy / distributed settings. Several constructions designed for special settings are different from the standard client-server model. The schemes in [43], [46] harness a trusted proxy to perform audit operations on behalf of the client, and/or to enable efficient updates. There are a few PDP/PoR designs that verify the integrity of the data replicated in distributed storage servers [49], [23].

Other application use cases. While PDP/PoR was originally proposed for integrity checks of remote data storage, they have been found useful in many other applications and settings. For example, proof-of-space [24], [38] uses PoR to develop an alternative to the traditional hash-based proof of work in blockchain. "Proof of data reliability" (or proof-of-replication) also harnesses PoR to verify not only data integrity but also redundancy for recovery in case of data corruption [27], [44]. PoR has also been used for digital fair exchange [5], [19].

II. Preliminaries

Notation. Let $\mathbb{F}$ be a finite field. Let $\mathbb{G}$ denote a cyclic group of prime order p and $\mathbb{Z}_p$ denote the ring of integers modulo p. $\langle \mathbf{x}, \mathbf{y} \rangle$ denotes the inner product between two vectors of the same length $\mathbf{x}$ and $\mathbf{y}$. We denote $[N] = \{1, \ldots, N\}$. Let $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_p^n$ be a vector and $\mathbf{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ be generators of $\mathbb{G}$, we denote $\mathbf{g}^{\mathbf{a}} = \prod_{i=1}^{n} g_i^{a_i}$. In expressions involving both polynomials and scalars, we write $f(X)$ instead of f to distinguish between the two; however, in contexts where it is clear that f is a polynomial, we simply write f for brevity. We denote F as the keyed pseudorandom function, where $k \leftarrow F.\mathsf{Gen}(1^\lambda)$ generates a PRF key k given security parameter λ and $y \leftarrow F(k, x)$ outputs an n-bit "random-looking" string y given PRF key k and input x.

A. Commitment Scheme

1) Polynomial Commitment: Polynomial commitment [32] permits a committer to commit to a polynomial in such a way that he can later reveal the evaluations of the polynomial at some evaluation points and prove that they correspond to the committed polynomial. Let $\mathbb{F}$ be a finite field and f be a polynomial on $\mathbb{F}$ with degree D. A polynomial commitment for $f \in \mathbb{F}^D[X]$ and $a \in \mathbb{F}$ is a tuple of PPT algorithms $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Eval}, \mathsf{Verify})$ as follows.

- $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D)$: Given a security parameter λ and a bound on the polynomial degree D, it generates public parameters pp.
- $\mathsf{cm} \leftarrow \mathsf{PC.Com}(f, \mathsf{pp})$: Given a polynomial $f \in \mathbb{F}^{D+1}[X]$, it computes a commitment cm with public parameter pp.
- $(y, \pi) \leftarrow \mathsf{PC.Eval}(f, \alpha, \mathsf{pp})$: Given an evaluation point $\alpha \in \mathbb{F}$, it computes $y = f(\alpha)$ and the proof π.
- $\{0, 1\} \leftarrow \mathsf{PC.Verify}(\mathsf{cm}, \alpha, y, \pi, \mathsf{pp})$: Given a commitment cm, an evaluation point α, an answer y, and a proof π, it outputs 1 if the evaluation is correct; otherwise it outputs 0.

Definition 1. PC satisfies the following properties.

- *Binding.* For any PPT adversary $\mathcal{A}$ such that $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D)$ and $(f_0, f_1) \leftarrow \mathcal{A}(\mathsf{pp})$, it holds that $\Pr\left[\mathsf{PC.Com}(f_0, \mathsf{pp}) = \mathsf{PC.Com}(f_1, \mathsf{pp}) \wedge f_0 \neq f_1\right] \leq \mathsf{negl}(\lambda)$
- *Completeness.* For any polynomial $f \in \mathbb{F}^{D+1}[X]$ and $\alpha \in \mathbb{F}$ such that $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D)$, $\mathsf{cm} \leftarrow \mathsf{PC.Com}(f, \mathsf{pp})$, and $(y, \pi) \leftarrow \mathsf{PC.Eval}(f, \alpha, \mathsf{pp})$, it holds that $\Pr\left[\mathsf{PC.Verify}(\mathsf{cm}, a, y, \pi, \mathsf{pp}) = 1\right] = 1$
- *Knowledge soundness.* For any PPT $\mathcal{A}$, $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D)$, there exists a PPT extractor $\mathcal{E}$ such that, given any tuple $(\mathsf{pp}, \mathsf{cm}^*)$ and the execution process of $\mathcal{A}$, $\mathcal{E}$ can extract $f \in \mathbb{F}^{D+1}[X]$ such that
$$\Pr\left[\begin{array}{c}(\pi^*, y^*, \alpha^*) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}) \\ f^* \leftarrow \mathcal{E}^{\mathcal{A}}(\mathsf{cm}^*, \mathsf{pp}) \\ \mathsf{PC.Verify}(\mathsf{cm}^*, \alpha^*, y^*, \pi^*, \mathsf{pp}) = 1 \\ f^*(\alpha^*) \neq y^*\end{array}\right] \leq \mathsf{negl}(\lambda)$$
- *Homomorphism.* A PC scheme is homomorphic if
$$\begin{aligned}\mathsf{PC.Com}(f) \boxplus \mathsf{PC.Com}(g) &= \mathsf{PC.Com}(f + g) \\ \mathsf{PC.Com}(f) \boxtimes c &= \mathsf{PC.Com}_r(c \cdot f)\end{aligned} \quad (1)$$
where $\boxplus, \boxtimes$ denotes addition and multiplication on the commitment space C, $c \in \mathbb{F}$ is a scalar.

DLP-based PC. We recall PC schemes, in which the $(D-1)$-degree polynomial is committed using a set of generators $(g_1, \ldots, g_D) \in \mathbb{G}$. Given a polynomial $f(X) = \sum_{i=1}^{D} a_i \cdot X^{i-1}$, we denote $\mathbf{a} = (a_1, \ldots, a_D)$ as the vector containing all coefficients of $f(X)$.

- *KZG Scheme [32]:* Kate et al. proposed an efficient PC scheme using bilinear mapping. Let $\mathbb{G}, \mathbb{G}_T$ be two groups of prime order p and $g \in \mathbb{G}$ be the generator. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be the bilinear map and $\mathsf{bp} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BilGen}(1^\lambda)$ denote the parameters generated for the bilinear map e.
  - $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D)$: Given a security parameter λ, and a bound on the polynomial degree D, it executes $\mathsf{bp} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BilGen}(1^\lambda)$ and outputs $\mathsf{pp} = (p, g, \mathbb{G}, e, \mathbb{G}_T, \mathbf{g})$, where $\mathbf{g} = (g, g^{\tau^1}, \ldots, g^{\tau^{D-1}})$.
  - $\mathsf{cm} \leftarrow \mathsf{PC.Com}(f, \mathsf{pp})$: It outputs $\mathbf{g}^{\mathbf{a}}$.
  - $(y, \pi) \leftarrow \mathsf{PC.Eval}(f, \alpha, \mathsf{pp})$: It outputs $y = f(\alpha)$ and $\pi = g^{q(\tau)}$, where $q(X) = \frac{f(X) - y}{X - \alpha}$.

- $\{0,1\} \leftarrow \mathsf{PC.Verify}(\mathsf{cm}, \alpha, y, \pi, \mathsf{pp})$: It outputs $\mathsf{e}(\mathsf{cm}/g^y, g) \stackrel{?}{=} \mathsf{e}(\pi, g^\tau/g^\alpha)$.

- *Inner Product Argument:* PC can also be derived from an inner product argument. We recall Bulletproofs [18] that proposes an efficient inner product proof system for the relation $\{(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, P \in \mathbb{G}, c \in \mathbb{Z}_p; \mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n) : P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}} \wedge c = \langle \mathbf{a}, \mathbf{b} \rangle\}$ where $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$ are public independent generators. To construct PC from the inner product argument, the polynomial coefficients can be treated as $\mathbf{a}$ while the evaluation point is treated as $\mathbf{b}$ in the above relation. The polynomial $f(X) = \sum_{i=0}^{D} a_i \cdot X^i$ is committed as $\mathbf{g}^{\mathbf{a}}$, where $\mathbf{g} = (g_0, \ldots, g_D)$ are public independent generators and $\mathbf{a} = (a_0, \ldots, a_D)$. We present in detail an efficient PC scheme based on the inner product argument in §V-B.

2) Vector Commitment: Vector commitment (VC) [22] permits a committer to commit to a vector in a way that some elements of the committed vector can be revealed and proven to belong to the original commitment (proof of membership). A VC for a message space $\mathcal{M}$ is a tuple of PPT algorithms $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{UpdCom}, \mathsf{Open}, \mathsf{Verify})$ as follows.

- $\mathsf{pp} \leftarrow \mathsf{VC.Setup}(1^\lambda, N)$: Given a security parameter λ and a bounded vector size N, it outputs public parameters pp.
- $\mathsf{cm} \leftarrow \mathsf{VC.Com}(\mathbf{x}, \mathsf{pp})$: Given a vector $\mathbf{x} \in \mathcal{M}^N$, it outputs a commitment cm.
- $\mathsf{cm}' \leftarrow \mathsf{VC.UpdCom}(\mathsf{cm}, i, \mathbf{x}[i], \mathbf{x}'[i], \mathsf{pp})$: Given a commitment cm, an index $i \in [N]$, an old element $\mathbf{x}[i]$ and a new element $\mathbf{x}'[i]$, it outputs an updated commitment cm'.
- $(y, \pi) \leftarrow \mathsf{VC.Open}(\mathbf{x}, i, \mathsf{pp})$: Given a vector $\mathbf{x}$, an index $i \in [N]$, it outputs $y = \mathbf{x}[i]$ and a proof π.
- $\{0,1\} \leftarrow \mathsf{VC.Verify}(\mathsf{cm}, y, i, \pi, \mathsf{pp})$: Given a vector commitment cm', an index $i \in [N]$, and a value y, a proof π, it outputs 1 if π is valid proof for which y is the i-th element of the vector committed to cm, and 0 otherwise.

Similar to PC, VC satisfies properties including *binding*, *completeness*, and *soundness*. Due to space constraints, we refer to [22] for their formal definition.

B. Error Correcting Code

Our construction uses the erasure code defined as follows.

Definition 2 (Erasure Codes). Denote Λ as a finite alphabet. An $(m, n, d)_\Lambda$ erasure code is a pair of PPT algorithms: encode: $\Lambda^n \rightarrow \Lambda^m$ and decode: $\Lambda^{m-d+1} \rightarrow \Lambda^n$, where $m > n \geq d$, such that as long as the number of erasures is bounded by $d-1$, decode can always recover the original data with probability 1. A code is the maximum distance separable (MDS), if $n + d = m + 1$.

ICC Encoding. We recall a $(2n, n, n)$-linear encoding scheme [41], which encodes n data blocks into $2n$ codeword blocks, in which any knowledge of n codeword blocks can be able to recover n original data blocks. To reduce the cost of recomputing the entire codeword for each time a data block is updated, Shi et al. proposed an Incrementally Constructible Code (ICC) [41], which permits updating only a small portion of the erasure code when a data block is updated. The idea is to build a hierarchical codeword $\mathbf{H}$ with $L = \lceil \log_2 n \rceil$ levels,

Parameters: Let $p = \alpha 2^n + 1$ denote a prime for some integer $\alpha \in \mathbb{N}$. Let $\mathbb{Z}_p^*$ denote $\mathbb{Z}_p \setminus \{0\}$ and $\hat{g}$ be a generator of $\mathbb{Z}_p^*$. Let $w = \hat{g}^\kappa \mod p$ be a $2n$-th primitive root of unity mod p.

$\mathsf{HAddB}(\mathbf{b}, t, \mathbf{H})$:
/ Suppose each $\mathbf{H}_\ell$ is of the form $\mathbf{H}_\ell := (\mathbf{X}_\ell, \mathbf{Y}_\ell)$, where $\mathbf{X}_\ell$ and $\mathbf{Y}_\ell$ each stores 2^ℓ codeword blocks.*/*
1. If $\mathbf{H}_0$ is empty:
2. $\quad \mathbf{X}_0 \leftarrow \mathbf{b}$ and $\mathbf{Y}_0 \leftarrow w^t \cdot \mathbf{b}$
3. Else:
4. $\quad$ Let $0, \ldots, \ell$ be successively full levels, and $\ell+1$ be first empty level
5. $\quad$ Call $\mathsf{HRebuildBX}(\ell, \mathbf{b})$ and $\mathsf{HRebuildBY}(\ell, w^t \cdot \mathbf{b})$

$\mathsf{HRebuildBX}(\ell, \mathbf{b})$:
/ $\mathsf{HRebuildBY}$ is analogue to $\mathsf{HRebuildBX}$ by replacing $\mathbf{X}$'s with $\mathbf{Y}$'s */*
1. $\tilde{\mathbf{X}}_1 \leftarrow \mathsf{mixB}(\mathbf{X}_0, \mathbf{b}, 0)$
2. For $i = 1$ to $\ell - 1$:
3. $\quad \tilde{\mathbf{X}}_{i+1} \leftarrow \mathsf{mixB}(\mathbf{X}_i, \tilde{\mathbf{X}}_i, i)$
4. Output $\mathbf{X}_\ell := \tilde{\mathbf{X}}_\ell$ and empty all $\mathbf{X}_0, \ldots, \mathbf{X}_{\ell-1}$

$\mathbf{H} \leftarrow \mathsf{mixB}(\mathbf{H}^0, \mathbf{H}^1, \ell)$:
1. Let $\nu = w^{n/2^\ell}$ be $2^{\ell+1}$-th primitive root of unity
2. For $i = 0$ to $2^\ell - 1$:
3. $\quad \mathbf{H}[i] \leftarrow \mathbf{H}^0[i] + \nu^i \cdot \mathbf{H}^1[i] \pmod p$
4. $\quad \mathbf{H}[i + 2^\ell] \leftarrow \mathbf{H}^0[i] - \nu^i \cdot \mathbf{H}^1[i] \pmod p$
5. Output $\mathbf{H}$

Fig. 1: ICC code [41].

where each level $\mathbf{H}_\ell$ contains $2^{\ell+1}$ erasure blocks of 2^ℓ most recently updated data blocks. Thus, the codeword $\mathbf{H}_\ell$ only needs to be computed every 2^ℓ updates. Figure 1 presents the algorithm to update the hierarchical codeword $\mathbf{H}$ when updating a data block $\mathbf{b}$ using the FFT algorithm.

Let t be an incremental value indicating the time when updating a block and $\mathbf{x}_\ell$ contains 2^ℓ blocks being updated recently, i.e., $\mathbf{x}_\ell$ contains blocks updated at time $t, t+1, \ldots, t+2^\ell-1$. Let $\psi_c(n)$ be a partial *bit-reversal function*, which reverses the least significant c bits of value n. For example, assume 3-bit representation, $\psi_3(1) = 4$ and $\psi_2(1) = 2$. Let $\pi_c(\mathbf{x})$ be a partial *bit-reversal permutation* function, where the element in $\mathbf{x}$ at index i is permuted to the index $\psi_c(i)$. Each $\mathbf{H}_\ell$ constructed in Figure 1 is of the following form

$$\mathbf{H}_\ell := \pi_\ell(\mathbf{v}_\ell) \times [\mathbf{F}_\ell \mid \mathbf{D}_{\ell,t}\mathbf{F}_\ell]$$

where $\mathbf{F}_\ell := \mathsf{vand}(w^{\frac{2n}{2^\ell}\cdot 2^0}, w^{\frac{2n}{2^\ell}\cdot 2^1}, \ldots, w^{\frac{2n}{2^\ell}\cdot 2^{\ell-1}})$ is a $2^\ell \times 2^\ell$ FFT Vandermonde matrix and $\mathbf{D}_{\ell,t} := \mathsf{diag}(w^{\psi_{\ell+1}(t)}, w^{\psi_{\ell+1}(t+1)}, \ldots, w^{\psi_{\ell+1}(t+n-1)})$ is a $2^\ell \times 2^\ell$ diagonal matrix.

Lemma 1. *Any $2^\ell \times 2^\ell$ submatrix of the generator matrix $\mathbf{G} := [\mathbf{F}_\ell \mid \mathbf{D}_{\ell,t}\mathbf{F}_\ell]$ is non-singular.*

III. Our Models

A. System Model

Our system model consists of a client and a server. The server provides data storage and access service to the client with integrity and retrievability guarantees, meaning that any portion of the client data is kept intact and retrievable (i.e., no deletion or corruption). A DPoR scheme is a tuple of PPT algorithms $(\mathsf{PSetup}, \mathsf{PRead}, \mathsf{PWrite}, \mathsf{PAudit})$ as follows.

- $(st, \mathcal{M}) \leftarrow \mathsf{PSetup}(1^\lambda, \mathbf{M}, N, \beta)$: Given a security parameter λ and a database $\mathbf{M}$ with N β-bit entries, it initializes the client state st and the server state $\mathcal{M}$.
- $\mathbf{b} \leftarrow \mathsf{PRead}(i)$: Given an index $i \in [N]$, it reads the database block as $\mathbf{b} \leftarrow \mathbf{M}[i]$. It outputs $\mathbf{b}$ or reject.

- $\mathsf{PWrite}(i, \mathbf{b}')$: Given an index $i \in [N]$, a data block $\mathbf{b}'$, it writes to the index i of the database as $\mathbf{M}[i] \leftarrow \mathbf{b}'$.
- $b \leftarrow \mathsf{PAudit}()$: It verifies if the server is maintaining $\mathbf{M}$ correctly so that all the blocks in $\mathbf{M}$ remain retrievable. It outputs a decision $b \in \{\mathsf{accept}, \mathsf{reject}\}$.

B. Threat and Security Models

In our system, the client is trusted. The server is untrusted and can behave maliciously. The server can deviate from the protocol to compromise the client's data authenticity and retrievability, for example, by discarding the latest updates from the user, modifying data content without being authorized to compromise data integrity, and/or deleting a portion of the user data to save storage space. Our security definition captures the *authenticity* and *retrievability* for an honest client in the presence of a malicious server as follows.

Definition 3 (Authenticity [20], [41]). Consider the following game $\mathsf{AuthGame}_{\mathcal{S}^*}(\lambda)$ between a malicious server $\mathcal{S}^*$ and a challenger as follows.

- The challenger initializes a copy of the honest client $\mathcal{C}$ and the honest server $\mathcal{S}$. $\mathcal{S}^*$ specifies an initial database $\mathbf{M}$. The challenger executes $(st, \mathcal{M}) \leftarrow \mathsf{PSetup}(1^\lambda, \mathbf{M}, N, \beta)$ on behalf of $\mathcal{C}$ and outputs $\mathcal{M}$ to both $\mathcal{S}^*$ and $\mathcal{S}$.
- At each time step $i \in [q]$, $\mathcal{S}^*$ adaptively specifies an operation $\mathsf{op}_i \in \{\mathsf{PRead}(j), \mathsf{PWrite}(j, \mathbf{b}'), \mathsf{PAudit}\}$ for some $j \in [N]$. The challenger executes the corresponding protocol indicated by op_i between $\mathcal{C}$ and $\mathcal{S}^*$, and passes every message from $\mathcal{C}$ to $\mathcal{S}$.
- If, at any execution time, the message given by $\mathcal{S}^*$ differs from that of $\mathcal{S}$, and $\mathcal{C}$ does *not* output reject, the adversary *wins* and the game outputs 1. Otherwise, it outputs 0.

We say that a DPoR scheme achieves *authenticity*, if for any PPT adversary $\mathcal{S}^*$, $\Pr[\mathsf{AuthGame}_{\mathcal{S}^*}(\lambda)] \leq \mathsf{negl}(\lambda)$.

Definition 4 (Retrievability [20], [41]). Consider the following game $\mathsf{ExtGame}_{\mathcal{S}^*,\mathcal{E}}(\lambda)$ between the malicious server $\mathcal{S}^*$, the extractor $\mathcal{E}$, and the challenger.

- **Initialize.** The challenger initializes a copy of the honest client $\mathcal{C}$. $\mathcal{S}^*$ specifies an initial database $\mathbf{M}$ with N β-bit blocks. The challenger executes $(st, \mathcal{M}) \leftarrow \mathsf{PSetup}(1^\lambda, \mathbf{M}, N, \beta)$ on behalf of $\mathcal{C}$ and outputs $\mathcal{M}$ to $\mathcal{S}^*$.
- **Query**. For each time step $i \in [q]$, $\mathcal{S}^*$ adaptively specifies an operation $\mathsf{op}_i \in \{\mathsf{PRead}(j), \mathsf{PWrite}(j, \mathbf{b}'), \mathsf{PAudit}\}$ for some $j \in [N]$. The challenger executes the corresponding protocol indicated by op_i between $\mathcal{C}$ and $\mathcal{S}^*$.
- **Challenge**. Let $\mathsf{Succ}(\mathcal{S}^*_{\mathsf{fin}}) \stackrel{\text{def}}{=} \Pr\left[\mathsf{PAudit}_{\mathcal{S}^*_{\mathsf{fin}},\mathcal{C}_{\mathsf{fin}}} = \mathsf{accept}\right]$ be the probability of executing a subsequent audit protocol between $\mathcal{S}_{\mathsf{fin}}$ and $\mathcal{C}_{\mathsf{fin}}$ over the random coins chosen by $\mathcal{C}_{\mathsf{fin}}$ that outputs accept.

1) Run $\mathbf{M}' \leftarrow \mathcal{E}^{\mathcal{S}^*_{\mathsf{fin}}}(\mathcal{C}_{\mathsf{fin}}, 1^N, 1^\lambda)$, where the extractor $\mathcal{E}$ gets blackbox rewinding access to the latest configuration of the malicious server at the end of query phase $\mathcal{S}^*_{\mathsf{fin}}$, and repeatedly executes PAudit protocol with $\mathcal{S}^*$ in $\mathsf{poly}(\lambda)$ times in attempt to extract database content $\hat{\mathbf{M}}$.
2) If $\hat{\mathbf{M}} \neq \mathbf{M}'$ and $\mathsf{Succ}(\mathcal{S}^*_{\mathsf{fin}}) \geq 1/\mathsf{poly}(\lambda)$, it outputs 1. Otherwise, it outputs 0.

We say that a DPoR scheme achieves *retrievability*, if there exists a PPT extractor $\mathcal{E}$ such that for any PPT adversary $\mathcal{S}^*$, $\Pr\left[\mathsf{ExtGame}_{\mathcal{S}^*,\mathcal{E}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$.

At an intuitive level, authenticity ensures that the client can always detect if the server deviates from the protocol description (e.g., by tampering with the protocol messages or input/output). On the other hand, retrievability ensures that the client data remains retrievable, in which if the adversarial server is in a state of passing an audit with a high probability, they must know the entire content of the client data. These two security properties are mandatory for any PoR construction with provable security against the malicious server.

IV. Homomorphic Authenticated Commitment

In this section, we construct a new homomorphic MAC for a homomorphic commitment. Recall that the audit process in most (D)PoR schemes (e.g., [20], [41], [31]) incurs checking the integrity of several data blocks for security against a malicious server. In this paper, we opt to use a MAC in our scheme to verify the integrity of the data being audited. Meanwhile, optimizing the audit overhead is one of our main goals. Therefore, given that random data blocks are checked per audit operation, it is mandatory for the MAC to be *homomorphic* so that the integrity of all audited blocks can be checked at once by a single MAC, thereby reducing the audit proof size and bandwidth cost. Therefore, towards enabling an efficient DPoR construction, our first step is to design a homomorphic MAC for a special message structure (i.e., commitment). We start with the formal definitions as follows.

A. Definitions

Definition 5 (Homomorphic MAC). A homomorphic MAC for homomorphic commitment over space C consists of a tuple of PPT algorithms $\Sigma = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{UpdState}, \mathsf{UpdTag}, \mathsf{Verify})$ as follows.

- $\kappa \leftarrow \Sigma.\mathsf{Setup}(1^\lambda)$: Given a security parameter λ, it generates a secret key $\kappa \in \mathsf{K}$.
- $\sigma \leftarrow \Sigma.\mathsf{Sign}_\kappa(\mathsf{cm}, st)$: Given a secret key $\kappa \in \mathsf{K}$, a commitment $\mathsf{cm} \in \mathsf{C}$ and a state $st \in \mathbb{F}$, it computes a tag $\sigma \in \mathsf{T}$ for cm under state st.
- $\sigma' \leftarrow \Sigma.\mathsf{Combine}\big((\sigma_1, \mathsf{cm}_1), \ldots, (\sigma_m, \mathsf{cm}_m)\big)$: Given m constants $(c_1, \ldots, c_m) \in \mathbb{F}$, and m tags $(\sigma_1, \ldots, \sigma_m) \in \mathsf{T}$ of commitments $(\mathsf{cm}_1, \ldots, \mathsf{cm}_m) \in \mathsf{C}$, it computes a tag $\sigma' \in \mathsf{T}$ for cm', where $\mathsf{cm}' = \boxplus_{i=1}^m c_i \boxtimes \mathsf{cm}_i$.
- $\tau \leftarrow \Sigma.\mathsf{UpdState}_\kappa\big((st_1, c_1), \ldots, (st_m, c_m), st'\big)$: Given a key $\kappa \in \mathsf{K}$, m states $(st_1, \ldots, st_m) \in \mathbb{F}$ and m constants $(c_1, \ldots, c_m) \in \mathbb{F}$, and a state $st' \in \mathbb{F}$, it computes an update token $\tau \in \mathsf{T}$ to update the tag of the aggregated commitment to state st'.

  $\sigma' \leftarrow \Sigma.\mathsf{UpdTag}(\tau, \sigma)$: Given an update token $\tau \in \mathsf{T}$ and a tag σ, it computes an updated tag $\sigma' \in \mathsf{T}$.
- $\{0, 1\} \leftarrow \Sigma.\mathsf{Verify}_\kappa\big(\mathsf{cm}, \sigma, (st_1, c_1), \ldots (st_m, c_m)\big)$: Given a secret key $\kappa \in \mathsf{K}$, a commitment $\mathsf{cm} \in \mathsf{C}$, a tag $\sigma \in \mathbb{G} \in \mathsf{T}$, m states $(st_1, \ldots, st_m) \in \mathbb{F}$ and m constants $(c_1, \ldots, c_m) \in \mathbb{F}$, it outputs accept (1) or reject (0).

Definition 6 (Unforgeability). Consider the following game $\mathsf{EUGame}_{\mathcal{A}}(\lambda)$ between the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$.

Setup. The challenger $\mathcal{C}$ samples a random key $\kappa \xleftarrow{\$} \mathsf{K}$, and initializes an empty list $\mathcal{L}$.

Query. $\mathcal{A}$ adaptively submits two types of the query to $\mathcal{C}$ as

- **Signing Query.** $\mathcal{A}$ adaptively submits to $\mathcal{C}$ a signing query (cm, st), where cm is a commitment, st is a state. $\mathcal{C}$ rejects if $st \in \mathcal{L}$. Otherwise, $\mathcal{C}$ computes $\sigma \leftarrow \Sigma.\mathsf{Sign}_{\kappa}(\mathsf{cm}, st)$ and sends σ to $\mathcal{A}$. $\mathcal{C}$ adds $(st; \mathsf{cm})$ to $\mathcal{L}$.
- **Update Query.** $\mathcal{A}$ adaptively submits to $\mathcal{C}$ an update query $((st_1, c_1), \ldots, (st_m, c_m), st')$. $\mathcal{C}$ rejects if $st' \in \mathcal{L}$ or if $st_j \notin \mathcal{L}$ for some $j \in [m]$. Otherwise, $\mathcal{C}$ computes $\tau \leftarrow \Sigma.\mathsf{UpdState}_{\kappa}((st_1, c_1), \ldots, (st_m, c_m), st')$ and sends τ to $\mathcal{A}$. $\mathcal{C}$ computes $\mathsf{cm}' = \prod_{j=1}^{m}(c_j \boxtimes \mathsf{cm}_j)$, where $\mathsf{cm}_j \leftarrow \mathcal{L}(st_j)$ and adds $(st'; \mathsf{cm}')$ to $\mathcal{L}$.

Output. $\mathcal{A}$ outputs a commitment cm^*, a tag σ^*, m states $(st_1^*, \ldots, st_m^*)$, and m constants $(c_1^*, \ldots, c_m^*)$. $\mathcal{A}$ *wins* and the game outputs 1 if (i) $\Sigma.\mathsf{Verify}_{\kappa}(\mathsf{cm}^*, \sigma^*, (st_1^*, c_1^*), \ldots, (st_m^*, c_m^*)) = 1$, (ii) $(c_1^*, \ldots, c_m^*)$ are not all zeros (trivial forgery), and (iii) either

1) $\exists st_i^* \notin \mathcal{L}$ for some $i \in [m]$, or
2) $st_i^* \in \mathcal{L}, \forall i \in [m]$ and $\mathsf{cm}^* \neq \boxplus_{i=1}^{m}(c_i \boxtimes \mathsf{cm}_i)$, where $\mathsf{cm}_i \leftarrow \mathcal{L}(st_i^*)$

We say that Σ achieves *unforgeability* if $\Pr[\mathsf{EUGame}_{\mathcal{A}}(\lambda)] \leq \mathsf{negl}(\lambda)$.

Intuitively, unforgeability ensures that an adversary (e.g., server) cannot generate a valid authenticated tag of a new commitment that is not in the form of a linear combination of some commitments, whose tags can be known by the adversary. This security is later needed in our DPoR construction to achieve authenticity in Definition 3 against a malicious server that may generate fake client data to bypass the audit.

B. Our MAC Scheme for DLP-based Commitment

We construct a new homomorphic MAC for the homomorphic polynomial commitment of the form $\mathbf{g}^{\mathbf{v}} \in \mathbb{G}$ that achieves unforgeability. Let $\mathsf{T}, \mathsf{C} = \mathbb{G}$, and h be a generator, we present the detailed algorithm in Figure 2.

Theorem 1 (Unforgeability of Σ). *Assuming F is a secure keyed PRF, our Σ scheme in Figure 2 is a secure homomorphic MAC by Definition 6. Specifically, let Adv be the advantage of $\mathcal{A}$ winning the above EUGame. For all homomorphic MAC adversaries $\mathcal{A}$, there exists a PRF adversary $\mathcal{A}'$ such that*

$$\mathsf{Adv}_{\mathsf{EUGame}}[\mathcal{A}, \Sigma] \leq \mathsf{Adv}_{\mathsf{PRF}}[\mathcal{A}', F] + (1/p).$$

Proof: See Appendix §A. ∎

V. Our Proposed Dynamic PoR

A. Generic Construction

We first present the generic construction of our scheme. We then instantiate it with two popular PC schemes including KZG [32] and inner product argument [18]. We first present the data structures of our scheme as follows.

$\kappa \leftarrow \Sigma.\mathsf{Setup}(1^{\lambda})$:
1. $\alpha \xleftarrow{\$} \mathbb{Z}_p^*$
2. $k \leftarrow F.\mathsf{Gen}(1^{\lambda})$
3. Output $\kappa := (\alpha, k)$

$\sigma \leftarrow \Sigma.\mathsf{Sign}_{\kappa}(\mathbf{g}^{\mathbf{v}}, st)$:
1. $r \leftarrow F(k, st)$
2. $\sigma \leftarrow (\mathbf{g}^{\mathbf{v}})^{\alpha} h^{r}$
3. Output σ

$\sigma \leftarrow \Sigma.\mathsf{Combine}((\sigma_1, c_1), \ldots (\sigma_m, c_m))$:
1. $\sigma' \leftarrow \prod_{i=1}^{m} \sigma_i^{c_i}$
2. Output σ'

$\tau \leftarrow \Sigma.\mathsf{UpdState}_{\kappa}((st_1, c_1), \ldots (st_m, c_m), st')$:
1. $r \leftarrow \sum_{i=1}^{m} c_i \cdot F(k, st)$
2. $r' \leftarrow F(k, st')$
3. $\tau \leftarrow h^{r'-r}$
4. Output τ

$\sigma' \leftarrow \Sigma.\mathsf{UpdTag}(\tau, \sigma)$:
1. $\sigma' \leftarrow \sigma \cdot \tau$
2. Output σ'

$\{0,1\} \leftarrow \Sigma.\mathsf{Verify}_{\kappa}(\mathbf{g}^{\mathbf{v}}, \sigma, (st_1, c_1), \ldots (st_m, c_m))$:
1. $r' \leftarrow \sum_{i=1}^{m} [c_i \cdot F(k, st_i)]$
2. $a \leftarrow (\mathbf{g}^{\mathbf{v}})^{\alpha}$ and $b \leftarrow h^{r'}$
3. **If** $a \cdot b = \sigma$ **then** output 1; **else** output 0

Fig. 2: Homomorphic MAC for DL-based commitment.

Data structures. Our scheme follows the same data structures in [41], which includes a raw buffer, a hierarchical log and an erasure-coded copy of the raw buffer.

- **Raw buffer.** To enable efficient read operations, we store N original data blocks in a raw buffer $\mathbf{U}$.
- **Erasure-coded copy of the raw buffer.** To ensure all the data blocks in $\mathbf{U}$ are retrievable and recoverable, we create the erasure code (denoted as $\mathbf{C}$) for the raw buffer $\mathbf{U}$.
- **Hierarchical log.** To support efficient update, we use a hierarchical structure $\mathbf{H}$ to instantiate the ICC codes. In $\mathbf{H}$, there are $L+1$ levels denoted as $(\mathbf{H}_0, \ldots, \mathbf{H}_L)$, where $L = \lceil \log_2 N \rceil$. Each $\mathbf{H}_\ell$ is an erasure code of 2^ℓ blocks being written most recently. $\mathbf{H}_0$ contains the erasure codeword for the most recently written block and the "age" of written blocks in $\mathbf{H}_\ell$ increases with ℓ.

Authenticated data structures. It is necessary to ensure the authenticity and freshness of all the blocks stored in the aforementioned data structures. In our scheme, we apply authentication techniques to each data structure. For the raw buffer $\mathbf{U}$, we use a VC technique to commit all the data blocks in $\mathbf{U}$ so that once a block is read from $\mathbf{U}$, its authenticity and freshness can be shown via a membership proof. For the hierarchical log $\mathbf{H}$ and erasure code $\mathbf{C}$, we use our MAC scheme in §IV to create the MAC tag for every codeword block (see below). We denote $\hat{\mathbf{C}}$ and $\hat{\mathbf{H}}$ as the MAC components of $\mathbf{C}$, and the hierarchical log $\mathbf{H}$, respectively.

We now present our generic DPOR construction. The detailed algorithms are given in Figure 3 and Figure 4.

Setup. The client first generates MAC key κ and public parameters $\mathsf{pp}, \mathsf{pp}'$ of the underlying PC and VC schemes, respectively (Figure 3, lines 2–5). Given a database DB with

$(st, \mathcal{M}) \leftarrow \mathsf{PSetup}(1^\lambda, \mathsf{DB}, N, \beta)$:

1. Initialize $t \leftarrow 0$, $\mathbf{H}_\ell \leftarrow \{\emptyset\}$ and $\hat{\mathbf{H}}_\ell \leftarrow \{\emptyset\}$ for $\ell \in \{0, \ldots, L\}$
2. Let $\mathsf{DB} := (\mathbf{b}_1, \ldots, \mathbf{b}_N)$
3. $\kappa = (\alpha, k) \leftarrow \Sigma.\mathsf{Setup}(1^\lambda)$
4. $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, \lfloor \beta / |\mathbb{F}| - 1 \rfloor)$
5. $\mathsf{pp}' \leftarrow \mathsf{VC.Setup}(1^\lambda, N)$
6. $\mathbf{U} \leftarrow (\mathbf{b}_1, \ldots, \mathbf{b}_N)$
7. $\mathsf{cm}' \leftarrow \mathsf{VC.Com}(\mathbf{U}, \mathsf{pp}')$
8. $\mathbf{C} := (\mathbf{c}_1, \ldots, \mathbf{c}_{2N}) \leftarrow \mathsf{encode}(\mathbf{U})$
9. **For** $i = 1$ to $2N$ **do**
10. $\quad \mathsf{cm}_i \leftarrow \mathsf{PC.Com}(\mathbf{c}_i, \mathsf{pp})$
11. $\quad st_i \leftarrow (L+1 || i || 0)$
12. $\quad \sigma_i \leftarrow \Sigma.\mathsf{Sign}_\kappa(\mathsf{cm}_i, st_i)$
13. $\hat{\mathbf{C}} \leftarrow (\sigma_1, \ldots, \sigma_{2N})$
14. $st \leftarrow (t, \kappa, \mathsf{cm}', \mathsf{pp}, \mathsf{pp}')$, $\mathcal{M} \leftarrow (\mathbf{U}, \mathbf{H}, \hat{\mathbf{H}}, \mathbf{C}, \hat{\mathbf{C}}, \mathsf{pp}, \mathsf{pp}')$
15. Output $(st, \mathcal{M})$

$\mathbf{b} \leftarrow \mathsf{PRead}(i)$:

16. $\mathcal{S}$ executes $(\mathbf{b}, \pi) \leftarrow \mathsf{VC.Open}(\mathbf{U}, i, \mathsf{pp}')$
17. $\mathcal{S} \rightarrow \mathcal{C}$: $(\mathbf{b}, \pi)$
18. $\mathcal{C}$ executes: If $\mathsf{VC.Verify}(\mathsf{cm}', \mathbf{b}, i, \pi, \mathsf{pp}') = 1$, output $\mathbf{b}$. Else reject.

$\mathsf{PWrite}(i, \mathbf{b})$:

19. $\mathcal{C}$ updates $\mathsf{cm}' \leftarrow \mathsf{VC.UpdCom}(\mathsf{cm}', i, \mathbf{U}[i], \mathbf{b}, \mathsf{pp}')$
20. $\mathcal{C} \rightarrow \mathcal{S}$: $\mathbf{b}$
21. $\mathcal{S}$ computes:
22. $\quad \mathbf{U}[i] \leftarrow \mathbf{b}$
23. $\quad$ Call $\mathsf{HAddB}(\mathbf{b}, t, \mathbf{H})$
24. C and $\mathcal{S}$ jointly compute:
25. $\quad$ Execute $\mathsf{HAddM}(st, \mathbf{g}^\mathbf{b}, t, \hat{\mathbf{H}})$
26. $\quad t \leftarrow t + 1$ ▷ Increment global timestamp
27. $\quad$ Every 2^H steps, call $\mathsf{CRebuild}()$

$\{0,1\} \leftarrow \mathsf{PAudit}()$:

28. $\mathcal{C}$ computes $c_{i,j} \xleftarrow{\$} [2^i - 1]$, $\rho_{i,j} \xleftarrow{\$} \mathbb{Z}_p^*$ for $0 \le i \le L+1$, $1 \le j \le \lambda$
29. $\mathcal{C} \rightarrow \mathcal{S}$: $\{c_{i,j}, \rho_{i,j}\}_{i=0,j=0}^{L+1,\lambda}$
30. $\mathcal{S}$ computes:
31. $\quad \mathbf{b} \leftarrow \sum_{i=0}^{L+1} \sum_{j=1}^{\lambda} \rho_{i,j} \cdot \mathbf{H}_i[c_{i,j}]$
32. $\quad$ Let $\sigma_{i,j} := \hat{\mathbf{H}}_i[c_{i,j}]$ for $0 \le i \le L+1$, $1 \le j \le \lambda$
33. $\quad \sigma' \leftarrow \Sigma.\mathsf{Combine}((\sigma_{0,1}, \rho_{0,1}), \ldots, (\sigma_{i,j}, \rho_{i,j}), \ldots, (\sigma_{L+1,\lambda}, \rho_{L+1,\lambda}))$
34. $\mathcal{S} \rightarrow \mathcal{C}$: $\sigma', \mathbf{g}^\mathbf{b}$
35. $\mathcal{C}$ computes:
36. $\quad st_{i,j} \leftarrow (i || c_{i,j} || t_i)$ for $0 \le i \le L+1$, $1 \le j \le \lambda$
37. $\quad b \leftarrow \Sigma.\mathsf{Verify}_\kappa(\mathbf{g}^\mathbf{b}, \sigma', (st_{0,1}, \rho_{0,1}), \ldots, (st_{i,j}, \rho_{i,j}), \ldots, (st_{L+1,\lambda}, \rho_{L+1,\lambda}))$
38. $\quad$ **If** $b = 0$ **then** output reject
39. $\quad x \xleftarrow{\$} \mathbb{Z}_p^*$
40. $\mathcal{C} \rightarrow \mathcal{S}$: x
41. $\mathcal{S}$ computes $(y, \pi) \leftarrow \mathsf{PC.Eval}(\mathbf{b}, x, \mathsf{pp})$
42. $\mathcal{S} \rightarrow \mathcal{C}$: y, π
43. $\mathcal{C}$ computes $b \leftarrow \mathsf{PC.Verify}(\mathbf{g}^\mathbf{b}, x, y, \pi, \mathsf{pp})$ and outputs b

Fig. 3: Our proposed Porla scheme.

$\mathsf{HAddM}(st, \mathbf{g}^\mathbf{b}, t, \hat{\mathbf{H}})$:

Let each $\hat{\mathbf{H}}_\ell$ be of form $\hat{\mathbf{H}}_\ell := (\hat{\mathbf{X}}_\ell, \hat{\mathbf{Y}}_\ell)$, where each $\hat{\mathbf{X}}_\ell, \hat{\mathbf{Y}}_\ell$ stores 2^ℓ corresponding MAC of codeword blocks in $\mathbf{H}_\ell$.

1. **If** $\hat{\mathbf{H}}_0$ is empty **then**
2. $\quad st \leftarrow (0 || 0 || t)$ ▷ *If $\hat{\mathbf{H}}_0$ is empty, use the latest timestamp*
3. $\quad \hat{\mathbf{X}}_0 \leftarrow \Sigma.\mathsf{Sign}_\kappa(\mathbf{g}^\mathbf{b}, st || 0)$
4. $\quad \hat{\mathbf{Y}}_0 \leftarrow \Sigma.\mathsf{Sign}_\kappa((\mathbf{g}^\mathbf{b})^{w^{\psi(t)}}, st || 1)$
5. **Else**
6. $\quad$ Let $0, \ldots, \ell$ be successively full lelvels, and $\ell+1$ be first empty level
7. $\quad st \leftarrow (0 || 1 || t-1)$ ▷ *$t-1$ is the time $\hat{\mathbf{H}}_0$ was last rebuilt*
8. $\quad \sigma_X \leftarrow \Sigma.\mathsf{Sign}_\kappa(\mathbf{g}^\mathbf{b}, st || 0)$
9. $\quad \sigma_Y \leftarrow \Sigma.\mathsf{Sign}_\kappa((\mathbf{g}^\mathbf{b})^{w^{\psi(t)}}, st || 1)$
10. $\quad$ Call $\mathsf{HRebuildMX}(\ell+1, \sigma_X)$ and $\mathsf{HRebuildMY}(\ell+1, \sigma_Y)$

$\mathsf{HRebuildMX}(\ell, \sigma)$:

/* $\mathsf{HRebuildMY}$ *is analogue to* $\mathsf{HRebuildMX}$ *by replacing* $\hat{\mathbf{X}}$*'s with* $\hat{\mathbf{Y}}$*'s* */

11. $\hat{\mathbf{X}}'_1 \leftarrow \mathsf{mixM}(\mathbf{X}_0, \sigma, 0)$
12. **For** $i = 1$ to $\ell - 1$ **do**
13. $\quad$ Let t_i, t_{i+1} be the timestamp level i and $i+1$ were last rebuilt.
14. $\quad$ If level $i+1$ is empty then $t_{i+1} \leftarrow t$
15. $\quad \hat{\mathbf{X}}'_{i+1} \leftarrow \mathsf{mixM}(\hat{\mathbf{X}}_i, \hat{\mathbf{X}}'_i, i, t_i, t_{i+1})$
16. Output $\hat{\mathbf{X}}_\ell \leftarrow \hat{\mathbf{X}}'_\ell$

$\hat{\mathbf{H}} \leftarrow \mathsf{mixM}(\hat{\mathbf{H}}^0, \hat{\mathbf{H}}^1, \ell, t_\ell, t_{\ell+1})$:

17. Let $\nu = w^{n/2^\ell}$ be $2^{\ell+1}$-th primitive root of unity
18. **For** $i = 0$ to $2^\ell - 1$ **do**
19. $\quad st_0 \leftarrow (\ell || i || t_\ell)$
20. $\quad st_1 \leftarrow (\ell || i + 2^\ell || t_\ell)$
21. $\quad$ **If** level $\ell+1$ is empty **then**
22. $\quad\quad st'_0 \leftarrow (\ell+1 || i || t_{\ell+1})$
23. $\quad\quad st'_1 \leftarrow (\ell+1 || i + 2^\ell || t_{\ell+1})$
24. $\quad$ **Else**
25. $\quad\quad st'_0 \leftarrow (\ell+1 || 2^\ell + i || t_{\ell+1})$
26. $\quad\quad st'_1 \leftarrow (\ell+1 || 2^\ell + i + 2^\ell || t_{\ell+1})$
27. $\quad \tau_0 \leftarrow \Sigma.\mathsf{UpdState}((st_0, 1), (st_1, \nu^i), st'_0)$
28. $\quad \hat{\mathbf{H}}[i] \leftarrow \hat{\mathbf{H}}^0[i] \cdot (\hat{\mathbf{H}}^1[i])^{\nu^i}$
29. $\quad \hat{\mathbf{H}}[i] \leftarrow \Sigma.\mathsf{UpdTag}(\hat{\mathbf{H}}[i], \tau_0)$
30. $\quad \tau_1 \leftarrow \Sigma.\mathsf{UpdState}((st_0, 1), (st_1, -\nu^i), st'_1)$
31. $\quad \hat{\mathbf{H}}[i + 2^\ell] \leftarrow \mathbf{H}^0[i] \cdot (\mathbf{H}^1[i])^{-\nu^i}$
32. $\quad \hat{\mathbf{H}}[i + 2^\ell] \leftarrow \Sigma.\mathsf{UpdTag}(\hat{\mathbf{H}}[i + 2^\ell], \tau_1)$
33. Output $\hat{\mathbf{H}}$

$\mathsf{CRebuild}()$:

34. **For** $i = 1$ to N **do**
35. $\quad \mathbf{b} \leftarrow \mathbf{U}[i]$
36. $\quad$ Execute $\mathsf{HAddB}(\mathbf{b}, t, \mathbf{C})$ and $\mathsf{HAddM}(st, \mathbf{g}^\mathbf{b}, t, \hat{\mathbf{C}})$

Fig. 4: Porla subroutines.

N β-bit blocks, the client stores all the blocks in the raw buffer $\mathbf{U}$ with N slots (line 6). The client then commits to $\mathbf{U}$ using a VC scheme (e.g., Merkle tree [36]) resulting in a commitment cm' (i.e., Merkle root), which is stored for later use to verify the authenticity of the retrieved block (line 7). Finally, the client builds the FFT-based erasure code of $\mathbf{U}$ as well as the authentication tag (lines 8–13). Specifically, for each codeword block in $\mathbf{C}$, the client commits to it using a PC scheme (line 10) and creates a MAC tag for the commitment (line 12).

Remark that our homomorphic MAC scheme requires a unique state (st) to compute the authentication tag for each commitment. This can be achieved by setting $st = (\mathsf{id} || i || t)$ (line 11), where id is the identifier of the data structure that the block belongs to, i is the block index, and t is a unique timestamp when the structure is (re)built (see below for details).

Read. Given an index $i \in [N]$, the server computes and returns $\mathbf{b} \leftarrow \mathbf{U}[i]$ along with a proof of membership (lines 16–17). The client verifies the proof against the commitment cm' and outputs $\mathbf{b}$ if the proof is valid (line 18).

Write. To write a block $\mathbf{b}$ to index i, the client updates the vector commitment cm to reflect the updated vector, where the old block at i is replaced by $\mathbf{b}$ (line 19). The client then sends the block to the server to update the raw buffer as $\mathbf{U}[i] \leftarrow \mathbf{b}$ (lines 20–22). Next, the client also sends the erasure code of $\mathbf{b}$ as well as its authentication tag to the server, which, in turn, will be placed at the top level of the hierarchical log, i.e., $\mathbf{H}_0$ and $\hat{\mathbf{H}}_0$ (lines 23, 25 in Figure 3 and lines 1–4 in Figure 4).

Remark that if the first L' levels $(\mathbf{H}_0, \ldots, \mathbf{H}_{L'-1})$ are full, the server will execute a *rebuilding process* (Figure 4, line 10) to merge and mix all the blocks in these levels and the updated

block into the next empty level $\mathbf{H}_{L'}$ as follows.

• *Rebuild* $\mathbf{H}$*:* We rebuild $\mathbf{H}$ using the FFT encoding scheme in §II-B. In our scheme, the block components are processed the same as in [41], while the MAC components will be computed differently and need further processing. Specifically, once a new codeword block is computed by FFT, we also need to compute its tag accordingly. Recall that $\hat{\mathbf{H}}_\ell[i]$ contains the MAC of (the commitment of) $\mathbf{H}_\ell[i]$, which is of the form:

$$\hat{\mathbf{H}}_\ell[i] = \left(\mathbf{g}^{\mathbf{H}_\ell[i]}\right)^{\alpha} \cdot h^{F(k, st_{\ell,i})}$$

where $st_{\ell,i}$ denotes the state from which $\hat{\mathbf{H}}_\ell[i]$ is created. Since the MAC is homomorphic, we can thus apply the FFT linear combination on the MAC tags in the next level $\ell+1$ (Figure 4, lines 27, 30) as

$$\begin{aligned}
\hat{\mathbf{H}}_{\ell+1}[i] &= \hat{\mathbf{H}}^0_\ell[i] \cdot (\hat{\mathbf{H}}^1_\ell[i])^{\nu^i} \\
&= \left(\mathbf{g}^{\mathbf{H}^0_\ell[i]}\right)^{\alpha} \cdot h^{F(k, st^0_{\ell,i})} \cdot \left(\mathbf{g}^{\mathbf{H}^1_\ell[i]}\right)^{\alpha \cdot \nu^i} \cdot h^{F(k, st^1_{\ell,i}) \cdot \nu^i} \\
&= \mathbf{g}^{(\mathbf{H}^0_\ell[i] + \nu^i \cdot \mathbf{H}^1_\ell[i]) \cdot \alpha} \cdot h^{F(k, st^0_{\ell,i}) + \nu^i \cdot F(k, st^1_{\ell,i})} \\
&= \left(\mathbf{g}^{\mathbf{H}_{\ell+1}[i]}\right)^{\alpha} \cdot h^{F(k, st^0_{\ell,i}) + \nu^i \cdot F(k, st^1_{\ell,i})} \\
\hat{\mathbf{H}}_{\ell+1}[i+N] &= \hat{\mathbf{H}}^0_\ell[i] \cdot \hat{\mathbf{H}}^1_\ell[i]^{-\nu^i} \\
&= \left(\mathbf{g}^{\mathbf{H}^0_\ell[i]}\right)^{\alpha} \cdot h^{F(k, st^0_{\ell,i})} \cdot \left(\mathbf{g}^{\mathbf{H}^1_\ell[i]}\right)^{-\nu^i \cdot \alpha} \cdot h^{-\nu^i \cdot F(k, st^1_{\ell,i})} \\
&= \mathbf{g}^{(\mathbf{H}^0_\ell[i] - \nu^i \cdot \mathbf{H}^1_\ell[i]) \cdot \alpha} \cdot h^{F(k, st^0_{\ell,i}) - \nu^i \cdot F(k, st^1_{\ell,i})} \\
&= \left(\mathbf{g}^{\mathbf{H}_{\ell+1}[i+N]}\right)^{\alpha} \cdot h^{F(k, st^0_{\ell,i}) - \nu^i \cdot F(k, st^1_{\ell,i})}
\end{aligned} \tag{2}$$

It is easy to see that $\hat{\mathbf{H}}_{\ell+1}[i]$ in (2) is the valid MAC of $\mathbf{H}_{\ell+1}[i]$ for any $i \in [2N]$. However, there is an issue: the state of $\hat{\mathbf{H}}_{\ell+1}[i]$ depends on the state of other tags at level ℓ. To verify $\hat{\mathbf{H}}_{\ell+1}[i]$, we need to know $st^0_{\ell,i}, st^1_{\ell,i}$. Given that $\hat{\mathbf{H}}_{\ell+1}$ will be later used to compute level $\ell+2$, the linear combination will be further expanded, where the state of blocks at deeper levels depends on all upper levels. This significantly increases the computation overhead to verify the MAC of an arbitrary block in the hierarchical log.

To address this issue, we update $\hat{\mathbf{H}}_{\ell+1}[i]$ such that it can be verified by its own state ($st_{\ell+1,i}$). Specifically, after computing (2), we convert $\hat{\mathbf{H}}_{\ell+1}[i]$ to the form of $\left(\mathbf{g}^{\mathbf{H}_{\ell+1}[i]}\right)^{\alpha} \cdot h^{F(k, st_{\ell+1,i})}$. This can be done with algorithms Σ.UpdState and Σ.UpdTag in Figure 2, given that $st_{\ell+1,i}$ is unique and has never been used previously. There are two cases for $\hat{\mathbf{H}}_{\ell+1}$:

- If level $\ell+1$ is currently empty, $\hat{\mathbf{H}}_{\ell+1}$ will be the new FFT-erasure code of this level. In this case, we set $st_{\ell+1,i} = (\ell+1 || i || t)$ for $i \in [2N]$, where t is the current timestamp (Figure 4, lines 21–23).
- Otherwise, level $\ell+1$ is full, meaning there currently exists an erasure code at this level (denoted as $\hat{\mathbf{H}}^0_{\ell+1}$). In this case, $\hat{\mathbf{H}}_{\ell+1}$ will be treated as $\hat{\mathbf{H}}^1_{\ell+1}$ in (2) to rebuild the next level $\hat{\mathbf{H}}_{\ell+2}$. To avoid state duplication, we set $st_{\ell+1,i} = (\ell+1 || i+2N || t_{\ell+1})$, where $t_{\ell+1}$ is the timestamp when the level $\ell+1$ was last rebuilt/empty (i.e., the index of codewords in $\hat{\mathbf{H}}_{\ell+1}$ starts after $\hat{\mathbf{H}}^0_{\ell+1}$) (lines 24–26).

Remark 1. *One can observe that the state st to update the MAC component in our scheme is* data-independent. *Therefore, it can be pre-computed in the setup phase or with a background process. Pre-computation permits the client to further save the computation cost of $\mathcal{O}(2^\ell)$ group operations and $\mathcal{O}(2^\ell)|\mathbb{G}|$ network bandwidth overhead when rebuilding level ℓ.*

Audit. Our audit is similar to other ECC-based PoR schemes, in which the client checks the authenticity and freshness of $\mathcal{O}(\lambda)$ random codeword blocks of each erasure code. To reduce the audit proof size, we make use of the random linear combination and verifiable polynomial evaluation techniques. Specifically, the server first aggregates all the random codeword blocks requested by the client (Figure 3, line 31) as

$$\mathbf{b} = \sum_{\ell=0}^{L+1} \sum_{i \in \mathcal{I}} \rho_{i,j} \cdot \mathbf{H}_\ell[i] \tag{3}$$

where $\rho_{i,j} \in \mathbb{Z}^*_p$ are the random scalars and $i \in \mathcal{I}$ are random block indices indicated by the client (line 28).

The server computes the commitment of the aggregated codeword block as $\mathbf{g}^{\mathbf{b}}$ as well as its MAC (line 33) as

$$\sigma = \prod_{\ell=0}^{L+1} \prod_{i \in \mathcal{I}} (\hat{\mathbf{H}}_\ell[i])^{\rho_{i,j}} \tag{4}$$

The client verifies the authenticity of $\mathbf{g}^{\mathbf{b}}$ based on its tag σ (line 37), and then attests to the server's knowledge of $\mathbf{b}$ by challenging the server to evaluate the polynomial that represents $\mathbf{b}$ (lines 39–43). Specifically, the client challenges a random point $a \xleftarrow{\$} \mathbb{Z}^*_p$ and the server computes $f(a) = \sum_{i=1}^{n} b_i \cdot a^{i-1}$, where $\mathbf{b} = (b_1, \ldots, b_n)$, as well as a proof of evaluation π. The client is convinced with the server knowledge if π is a valid proof of the polynomial evaluation $f(a)$. Due to the random linear combination, knowing $\mathbf{b}$ implies the server knowledge of all individual $\mathbf{H}_\ell[i]$ being challenged.

B. Instantiations

We instantiate our generic construction with two polynomial commitment schemes including KZG [32] and Bulletproofs [18]. For KZG, we use the original KZG scheme for commitment, evaluation, and verification. We denote our KZG-based DPoR scheme as $\mathsf{Porla}_{\mathsf{kzg}}$. For Bulletproofs, we present a more simplified version of the Inner Product Argument (IPA) for verifiable polynomial commitment. We denote our IPA-based DPoR scheme as $\mathsf{Porla}_{\mathsf{ipa}}$. We note that our instantiations offer different security assumptions and efficiency trade-offs. Specifically, in the $\mathsf{Porla}_{\mathsf{kzg}}$ scheme, we can assume that the trusted client can have access to the trapdoor τ that is used to generate a common reference string in the KZG setup phase. This trapdoor permits the client to compute the commitment during data update faster with multi-multiplication group operations instead of multi-exponentiation. This advantage requires the trapdoor to be kept secret and never leaked out as a trade-off. We discuss the cost difference between two instantiations in more detail in §VII-C.

Polynomial Commitment via Inner Product Argument. We present a verifiable polynomial evaluation based on the original inner product argument proposed by Bulletproofs [18]. Let x be an evaluation point and $\mathbf{a} = (a_0, \ldots, a_D)$ be a vector containing the coefficients of polynomial $f(X) = \sum_{i=0}^{D} a_i \cdot X^i$. Let $\mathbf{g} = (g_0, \ldots, g_{D+1})$ be public independent generators. The verifiable polynomial evaluation is a proof system for the following relation $\{(\mathbf{g} \in \mathbb{G}^{D+1}, P \in \mathbb{G}, y \in \mathbb{Z}_p, x \in \mathbb{Z}_p; \mathbf{a} \in \mathbb{Z}^{D+1}_p) : P = \mathbf{g}^{\mathbf{a}} \wedge y = \langle \mathbf{a}, \mathbf{x} \rangle\}$ where $\mathbf{x} = (x^0, x^1, \ldots, x^D)$. For simplicity, we present the interactive version of the proof for the above relation in Figure 5. It is straightforward to make the proof non-interactive using Fiat-Shamir transformation [26].

Input: $(\mathbf{g}, P = \mathbf{g}^{\mathbf{a}}, \mathbf{a}, x)$
$\mathcal{P}$'s input: $(\mathbf{g}, \mathbf{a})$
$\mathcal{V}$'s input: $(\mathbf{g}, P, x)$
Output: $\{c = \langle \mathbf{a}, \mathbf{x} \rangle$, where $\mathbf{x} = (x^0, \ldots, x^D)$ if $\mathcal{V}$ accepts, or $\perp$ if rejects$\}$

1. $\mathcal{V} \to \mathcal{P}$: x
2. $\mathcal{P}$ computes:
3. $\quad \mathbf{x} = (x^0, \ldots, x^D)$
4. $\quad c \leftarrow \langle \mathbf{a}, \mathbf{x} \rangle$
5. $\mathcal{P} \to \mathcal{V} : c$
6. $\mathcal{V}$: $\alpha \xleftarrow{\$} \mathbb{Z}_p^*$
7. $\mathcal{V} \to \mathcal{P}$: α
8. $\quad P' \leftarrow P \cdot u^{\alpha \cdot c}$
9. Execute protocol below on **Input** $(\mathbf{g}, u^{\alpha}, P', \mathbf{x}; \mathbf{a})$

Input: $(\mathbf{g} \in \mathbb{G}^n, u, P \in \mathbb{G}, \mathbf{x} \in \mathbb{Z}_p^{D+1}; \mathbf{a} \in \mathbb{Z}_p^{D+1})$
$\mathcal{P}$'s input: $(\mathbf{g}, \mathbf{a}, \mathbf{x}, c)$
$\mathcal{V}$'s input: $(\mathbf{g}, u, P, \mathbf{x})$

1. **If** $n = 1$ **then**
2. $\quad \mathcal{P} \to \mathcal{V}$: $a \in \mathbb{Z}_p$
3. $\quad \mathcal{V}$ computes $c = a \cdot x$ and check if $P \stackrel{?}{=} g^a u^c$. If yes, $\mathcal{V}$ accepts; otherwise $\mathcal{V}$ rejects.
4. **Else**
5. $\quad \mathcal{P}$ computes:
6. $\quad\quad D' = D/2$
7. $\quad\quad c_L = \langle \mathbf{a}_{[:D']}, \mathbf{x}_{[D':]} \rangle$
8. $\quad\quad c_R = \langle \mathbf{a}_{[D':]}, \mathbf{x}_{[:D']} \rangle$
9. $\quad\quad L = \mathbf{g}_{[D':]}^{\mathbf{x}_{[:D']}} u^{c_L} \qquad R = \mathbf{g}_{[:D']}^{\mathbf{a}_{[D':]}} u^{c_R}$
10. $\quad \mathcal{P} \to \mathcal{V} : L, R$
11. $\quad \mathcal{V} : z \xleftarrow{\$} \mathbb{Z}_p^*$
12. $\quad \mathcal{V} \to \mathcal{P} : z$
13. $\quad$ Both $\mathcal{P}$ and $\mathcal{V}$ compute:
14. $\quad\quad \mathbf{g}' = \mathbf{g}_{[:D']}^{z^{-1}} \odot \mathbf{g}_{[D':]}^{z}$
15. $\quad\quad \mathbf{x}' = \mathbf{x}_{[D':]} \cdot z + \mathbf{x}_{[:D']} \cdot z^{-1}$
16. $\quad \mathcal{P}$ computes: $\mathbf{a}' = \mathbf{a}_{[:D']} \cdot z + \mathbf{a}_{[D':]} \cdot z^{-1}$
17. $\quad \mathcal{V}$ computes: $P' = L^{z^2} \cdot P \cdot R^{z^{-2}}$
18. $\quad$ Recursively repeat this protocol on **Input**$(\mathbf{g}', u, P', \mathbf{x}'; \mathbf{a}')$

Fig. 5: IPA-based Verifiable Polynomial Evaluation.

C. Security Analysis

Theorem 2. Porla *satisfies authenticity by Definition 3.*

Theorem 3. Porla *satisfies retrievability by Definition 4.*

We present the proofs in Appendices §B and §C.

D. Efficiency

We analyze the efficiency of our proposed scheme. Let λ, β, N be the security parameter, the data block size, and the number of data blocks in the database, respectively.

The read operation incurs transmitting a data block and a membership proof, which incurs a bandwidth cost of $\beta + |\pi_{\mathsf{vc}}|$. The client and server computation depends on the complexity of membership proof and verification of the underlying VC scheme. For example, with the Merkle tree [36], $|\pi_{\mathsf{vc}}| = \mathcal{O}(\lambda \log N)$, and the client incurs $\mathcal{O}(\log N)$ hash invocations.

For the write operation, the client receives a membership proof to update the vector commitment with the new data block. In total, the client bandwidth incurs $\beta + |\pi_{\mathsf{vc}}|$ bandwidth overhead. The client incurs the cost to update the vector commitment to capturing the new data block in the raw buffer, and the cost of $\mathcal{O}(\beta)$ group operations to compute a polynomial commitment for the new block. On the other hand, the server rebuilds some level of the hierarchical code. At each level ℓ, the server computes FFT codeword for 2^ℓ data blocks and 2^ℓ MAC, which incurs $\mathcal{O}(2^\ell \cdot \beta)$ field and $\mathcal{O}(2^\ell)$ group operations. Since each level ℓ is rebuilt every 2^ℓ write operations, the amortized rebuilding cost per write is $\mathcal{O}(\beta \log N)$ field and $\mathcal{O}(\log N)$ group operations.

For audit operation, the client generates $\mathcal{O}(\lambda \log N)$ random indices and random scalars, which can be sent with $\mathcal{O}(\lambda)$ bandwidth cost via a PRF seed. The client receives a MAC tag, a commitment of aggregated block, and a proof of the polynomial evaluation, which incurs a total of $\mathcal{O}(\lambda + |\pi|)$ bandwidth cost, where π is the proof size of the underlying PC scheme. Specifically, for the KZG scheme, $|\pi| = 1|\mathbb{G}|$ and the client incurs $\mathcal{O}(1)$ group operations and 1 pairing, while the server incurs $\mathcal{O}(\beta)$ group operations. For IPA-based PC, $|\pi| = \mathcal{O}(\log \beta)$ and the client/server overhead is $\mathcal{O}(\beta)$ group operations. The client incurs $\mathcal{O}(1)$ group operations to verify the MAC. Before proving the polynomial evaluation, the server performs a random linear combination on the data blocks and MAC components, and commits to the aggregated block, which incurs $\mathcal{O}(\beta \cdot \lambda \log N)$ field operations and $\mathcal{O}(\lambda \log N + \beta)$ group operations, respectively. In total, the audit proof size is $3|\mathbb{G}|$ for $\mathsf{Porla}_{\mathsf{kzg}}$ scheme (one commitment, one MAC tag, and one proof), and $\mathcal{O}(\log \beta)|\mathbb{G}|$ for $\mathsf{Porla}_{\mathsf{ipa}}$ scheme.

For the storage, the client only needs to store an $\mathcal{O}(\lambda)$-bit master key and the timestamp as the global counter. The server stores FFT codewords, which incurs $\mathcal{O}(N\beta)$ overhead, while the authentication component costs $\mathcal{O}(\lambda N)$ overhead.

VI. Public Auditability

In this section, we show how to enable public audit in Porla. The high-level idea is to somehow permit a public auditor to obtain the correct polynomial commitments of the challenged codeword blocks so that the server's knowledge of codeword blocks can be attested via a verifiable polynomial evaluation and a random linear combination. We achieve this by using VC technique at the server side, in which the server commits to a vector containing the polynomial commitments of codeword blocks, and then later proves the membership of polynomial commitments (challenged by a public auditor) corresponding to the committed vector.

For each level in $\mathbf{H}$ (or $\mathbf{C}$), suppose $\mathbf{v}_1, \ldots, \mathbf{v}_{n'}$ are the codeword blocks, and $\mathbf{g}^{\mathbf{v}_1}, \ldots, \mathbf{g}^{\mathbf{v}_{n'}}$ are their corresponding commitments. Let $\mathbf{x} := (\mathbf{g}^{\mathbf{v}_1}, \ldots, \mathbf{g}^{\mathbf{v}_{n'}})$, the server commits to $\mathbf{x}$ as $\mathsf{cm}' \leftarrow \mathsf{VC.Com}(\mathbf{x}, \mathsf{pp}')$, where $\mathsf{pp}' \leftarrow \mathsf{VC.Setup}(1^\lambda)$. The public audit protocol happens as follows.

1) For each level in $\mathbf{H}$ (or $\mathbf{C}$), the auditor samples $t = \mathsf{poly}(\lambda)$ random indices. Let $(i_1, \ldots, i_n)$ be the selected indices, where $n = t \cdot (1 + \log N)$. The auditor sends $(i_1, \ldots, i_n)$ to the server[1].
2) For each $j \in [n]$, the server executes $(\mathbf{g}^{\mathbf{v}_{i_j}}, \pi'_{i_j}) \leftarrow \mathsf{VC.Open}(\mathsf{cm}', i_j, \mathsf{pp}')$ and sends $(\mathbf{g}^{\mathbf{v}_{i_j}}, \pi'_{i_j})$ to the auditor.
3) For each $j \in [n]$, the auditor verifies $b \leftarrow \mathsf{VC.Verify}(\mathsf{cm}', i_j, \mathbf{g}^{\mathbf{v}_{i_j}}, \pi'_{i_j}, \mathsf{pp}')$. If all are valid, the auditor samples a random scalar ρ and computes $\mathbf{g}^{\mathbf{v}} = \mathbf{g}^{\sum_{j=1}^{n} \mathbf{v}_{i_j} \rho^{i_j}}$.
4) The server executes $(y, \pi) \leftarrow \mathsf{PC.Eval}(f, \alpha, \mathsf{pp})$, where α is auditor's random challenge, and $f(X) = \sum_{i=1}^{D} \mathbf{v}[i] \cdot X^{i-1}$, and returns (y, π) to the auditor. The auditor executes $b \leftarrow \mathsf{PC.Verify}(\mathbf{g}^{\mathbf{v}}, \alpha, y, \pi, \mathsf{pp})$ and outputs b.

[1] To reduce bandwidth, the auditor can simply send a PRF seed s.

TABLE II: Comparison of our Porla framework over state-of-the-art (public audit setting).

Scheme	Write			Public Audit		
	Bandwidth	(Private) client cost	Server cost	Proof size	Verification cost	Proving cost
SSP13 [41]	$(1+\epsilon)(\beta+\lvert\pi_{\mathsf{vc}}\rvert)$ $+\mathcal{O}(\lambda\log N)$	$(1+\epsilon)\mathcal{U}_{\mathsf{vc}}+$ $\mathcal{O}(\lambda\beta)\mathbb{F}+\mathcal{O}(\log N)\mathsf{E}$	$1\mathcal{C}_{\mathsf{vc}}+(1+\epsilon)\mathcal{U}_{\mathsf{vc}}$ $+\mathcal{O}(\beta\log N)\mathbb{F}$	$\mathcal{O}((\beta+\lvert\pi_{\mathsf{vc}}\rvert)\lambda\log N)$	$\mathcal{O}(\beta\lambda\log N)\mathcal{V}_{\mathsf{vc}}$	$\mathcal{O}(\beta\lambda\log N)\mathcal{P}_{\mathsf{vc}}$
ADJ+21 [7]	$\beta+\lvert\pi_{\mathsf{vc}}\rvert$	$1\mathcal{U}_{\mathsf{vc}}$	$1\mathcal{U}_{\mathsf{vc}}$	$\mathcal{O}(\lambda\sqrt{\beta N})$	$\mathcal{O}(\sqrt{\beta N})\mathbb{G}$	$\mathcal{O}(N\beta)\mathbb{F}$
Our $\mathsf{Porla}_{\mathsf{kzg}}$	$\beta+$ $(1+\epsilon)\lvert\pi_{\mathsf{vc}}\rvert$ $+(1+\epsilon)\lvert\mathbb{G}\rvert$	$(1+\epsilon)\mathcal{U}_{\mathsf{vc}}+\mathcal{O}(\beta)\mathbb{G}$	$1\mathcal{C}_{\mathsf{vc}}+(1+\epsilon)\mathcal{U}_{\mathsf{vc}}$ $+\mathcal{O}(\beta\log N)\mathbb{F}$ $+\mathcal{O}(\log N)\mathbb{G}$	$\mathcal{O}(\lambda\log N)\lvert\pi_{\mathsf{vc}}\rvert$ $+\mathcal{O}(\lambda\log N)\lvert\mathbb{G}\rvert$	$\mathcal{O}(\lambda\log N)\mathcal{V}_{\mathsf{vc}}$ $+\mathcal{O}(\lambda\log N)\mathbb{G}+1\mathsf{e}$	$\mathcal{O}(\lambda\log N)\mathcal{P}_{\mathsf{vc}}+$ $\mathcal{O}(\beta\lambda\log N)\mathbb{F}+$ $\mathcal{O}(\lambda\log N+\beta)\mathbb{G}$
Our $\mathsf{Porla}_{\mathsf{ipa}}$				$\mathcal{O}(\lambda\log N)\lvert\pi_{\mathsf{vc}}\rvert+2\lvert\mathbb{F}\rvert$ $+\mathcal{O}(\lambda\log N+\log\beta)\lvert\mathbb{G}\rvert$	$\mathcal{O}(\lambda\log N)\mathcal{V}_{\mathsf{vc}}$ $+\mathcal{O}(\lambda\log N+\beta)\mathbb{G}$	

CKW17 [20] does not support public audit. $\mathcal{P}_{\mathsf{vc}}, \mathcal{V}_{\mathsf{vc}}$ denote the cost of proving and verifying a membership of the underlying VC scheme, respectively. $\mathcal{C}_{\mathsf{vc}}$ denotes the cost to create a commitment for a vector with $\mathcal{O}(\log N)$ elements, and $\mathcal{U}_{\mathsf{vc}}$ denotes the cost to update the vector commitment.

In this public audit scheme, the server sends $\mathcal{O}(\lambda\log N)$ commitments and a membership proof for each of them in the vector commitment. Thus, the total proof size is $\mathcal{O}(\lambda\log N(\lambda+\lvert\pi_{\mathsf{vc}}\rvert)+\lvert\pi_{\mathsf{pc}}\rvert)$, where $\lvert\pi_{\mathsf{vc}}\rvert$ is the size for membership proof, $\lvert\pi_{\mathsf{pc}}\rvert$ is the proof size of the polynomial evaluation. The auditor cost is $\mathcal{O}(\lambda\log N)\mathcal{V}_{\mathsf{vc}}+\mathcal{O}(\lambda\log N)\mathbb{G}+\mathcal{V}_{\mathsf{pc}}$ and the server time is $\mathcal{O}(\lambda\log N)\mathcal{P}_{\mathsf{pc}}+\mathcal{O}(\beta\lambda\log N)\mathbb{F}+\mathcal{O}(\lambda\log N)\mathbb{G}+\mathcal{P}_{\mathsf{pc}}$, where $\mathcal{P}_{\mathsf{pc}}, \mathcal{V}_{\mathsf{pc}}$ (resp. $\mathcal{P}_{\mathsf{vc}}, \mathcal{V}_{\mathsf{vc}}$) are the overhead of proving and verifying a polynomial evaluation (resp. a membership), respectively. Note that the proof size can be reduced to $\mathcal{O}(\lambda^2\log N+\lvert\pi_{\mathsf{pc}}\rvert)$ using an aggregatable VC (e.g., [28]) that permits membership proof of multiple elements via a single proof at the cost of a linear proving time.

Updated rebuilding process. Since the vector commitment is computed by the server, we need to ensure that all the elements in the committed vector are correct and consistent with the codeword blocks after each data update. This is achieved by having the private client (i.e., data owner) check the authenticity and consistency of $\mathcal{O}(\lambda)$ random elements in the vector committed by the server. Note that the vector commitment is computed only during the rebuilding process. Thus, when rebuilding level $\mathbf{H}_\ell$ (or $\mathbf{C}$), the following procedure happens:

1) The server performs rebuilding on the codeword blocks and their MAC components as usual, and computes the commitment of the new codeword blocks. Let $\mathbf{v}_1,\ldots,\mathbf{v}_n$ be the new codeword blocks and $\mathbf{g}^{\mathbf{v}_1},\ldots,\mathbf{g}^{\mathbf{v}_{n'}}$ be their corresponding commitments. Let $\mathbf{x}:=(\mathbf{g}^{\mathbf{v}_1},\ldots,\mathbf{g}^{\mathbf{v}_{n'}})$, the server commits to $\mathbf{x}$ as $\mathsf{cm}'\leftarrow\mathsf{VC.Com}(\mathbf{x},\mathsf{pp}')$.
2) The private client challenges $t=\mathsf{poly}(\lambda)$ random elements in the committed vector. Let $(i_1,\ldots,i_t)$ be the challenged indices. For each $j\in[t]$, the server computes $(\mathbf{g}^{\mathbf{v}_{i_j}},\pi_{i_j})\leftarrow\mathsf{VC.Open}(\mathsf{cm}',i_j,\mathsf{pp}')$ and sends $(\mathbf{g}^{\mathbf{v}_{i_j}},\pi_{i_j})$ to the client. The server also performs a random linear combination on the MAC of the challenged commitments as $\sigma\leftarrow\Sigma.\mathsf{Combine}(\sigma_{i_1},\rho^{i_1}),\ldots,(\sigma_{i_t},\rho^{i_t}))$, where σ_{i_j} is the MAC of $\mathbf{g}^{\mathbf{v}_{i_j}}$ and ρ is a random scalar challenged by the client. The server sends σ to the client.
3) The client computes $b_{i_j}\leftarrow\mathsf{VC.Verify}(\mathsf{cm},i_j,\mathbf{g}^{\mathbf{v}_{i_j}},\pi_{i_j},\mathsf{pp}')$ for each $j\in[t]$. If $b_{i_j}=1$ for all $j\in[t]$, the client next checks $b\leftarrow\Sigma.\mathsf{Verify}_\kappa(\mathbf{g}^{\mathbf{v}},\sigma,(st_{i_1},\rho^{i_1}),\ldots,(st_{i_t},\rho^{i_t}))$, where $\mathbf{g}^{\mathbf{v}}:=\prod_{i=1}^{t}\mathbf{g}^{\mathbf{v}_{i_j}\cdot\rho^{i_j}}$. The former verifies the membership of individual $\mathbf{g}^{\mathbf{v}_{i_j}}$ in the committed vector, while the latter verifies their authenticity against the MAC.

Updated write cost. For each level ℓ, the client downloads λ polynomial commitments and verifies their membership against the committed vector. Since level ℓ is rebuilt after every 2^ℓ writes, it incurs a small amount $\epsilon=\frac{\lambda\log N}{N}$ of extra VC processing overhead to the private audit scheme in terms of bandwidth and client/server time. Specifically, the amortized client bandwidth per write is $\mathcal{O}(\beta+(1+\epsilon)\lvert\pi_{\mathsf{vc}}\rvert+\epsilon\lvert\mathbb{G}\rvert)$. The client time is $(1+\epsilon)\mathcal{U}_{\mathsf{vc}}+\mathcal{O}(\beta)\mathbb{G}$ and the server time is $1\mathcal{C}_{\mathsf{vc}}+(1+\epsilon)\mathcal{U}_{\mathsf{vc}}+\mathcal{O}(\beta\log N)\mathbb{F}+\mathcal{O}(\log N)\mathbb{G}$, where $\mathcal{C}_{\mathsf{vc}}$ is the complexity to create a VC commitment for a vector of size $\mathcal{O}(\log N)$ (for the membership proof per hierarchical level).

Theorem 4. *Our public audit variant satisfies authenticity by Definition 3 and retrievability by Definition 4.*

Proof: Due to the security of VC, the public auditor obtains correct commitments of the challenged codeword blocks. The rest follows the proof of private audit in Appendix §C. ■

VII. Experimental Evaluation

A. Implementation

We fully implemented all our proposed techniques in `C++` consisting of approximately 4,000 lines of code. In our implementation, we used standard cryptographic libraries, including `libNTL` [42] for modulo arithmetic, `libsecp256k1` [48] for elliptic curve operations in $\mathsf{Porla}_{\mathsf{ipa}}$ scheme. We implemented the verifiable polynomial evaluation based on the Bulletproofs' IPA from scratch. For $\mathsf{Porla}_{\mathsf{kzg}}$ scheme, we used `libgnark-crypto` library [16] for `BN254` curve to implement KZG polynomial commitment. We used `libzeromq` library [4] to implement network communication between the client and the server. Our implementation is available at https://github.com/vt-asaplab/porla.

Handling FFT Modulo over the Exponent. In our schemes, we implemented standard curves (i.e., `secp256k1` and `BN254`) for group operations. Since the order of these groups is not in the form of Proth prime p required by the FFT erasure code, our implementation handles the modulo p over the exponent when updating the MAC of polynomial commitment of new FFT codeword blocks during the rebuilding process as follows. Let q be the group order and $l=\mathsf{lcm}(p,q)$ be the least common multiple of p and q. For each intermediate level that is not the last level in the rebuilding process, the server computes the FFT codeword blocks modulo l (rather than modulo p). At the final level, for each modulo-l codeword block $\mathbf{c}$, the server computes the final modulo-p codeword block $\mathbf{v}=\mathbf{c}\pmod p$ and its corresponding alignment $\mathbf{g}^{\mathbf{v}-\mathbf{c}\pmod q}$ (where the multi-exponentiation is implemented with Pippenger's algorithm [37]). The alignment is needed for the client to correctly check the MAC of commitment of the FFT codeword blocks in the audit phase. During the audit, the server aggregates the alignment components of requested

blocks and sends the aggregation to the client, which incurs an extra 33-byte bandwidth overhead.

B. Experimental Configuration

Hardware and network setting. For the client, we used a laptop with an Intel i7-6820HQ CPU @ 2.7 GHz and 16 GB RAM. We used an Amazon EC2 `c6i.8xlarge` instance as the server, which is equipped with a 16-core Intel Xeon Platinum 8375C CPU @ 2.90GHz, 64 GB RAM, and 16 TB SSD in all experiments. The network bandwidth between the client and server is 217 Mbps with 7 ms round-trip latency.

Dataset. We used synthetic datasets of sizes ranging from 1 GB to 2 TB containing $N = 2^{12}$ to $N = 2^{29}$ dummy data blocks with different block sizes (i.e., $\beta \in \{4\,\text{KB}, 32\,\text{KB}, 256\,\text{KB}\}$).

Counterpart comparison and parameter choice. In this experiment, we compared the performance of our proposed technique with state-of-the-art DPoR schemes including SSP13 [41], CKW17 [20] , and ADJ+21 [7]. We also compared with a (standard) static PoR scheme (i.e., [31]) to demonstrate the cost difference between dynamic and static PoR constructions. Finally, we compared with a baseline (or trivial) proof of retrievability approach that transmits the entire database and verifies the checksum of each data block to demonstrate the effectiveness of (D)PoR schemes proposed in the literature (including ours). We selected the parameters for all the schemes to achieve 128-bit security as follows.

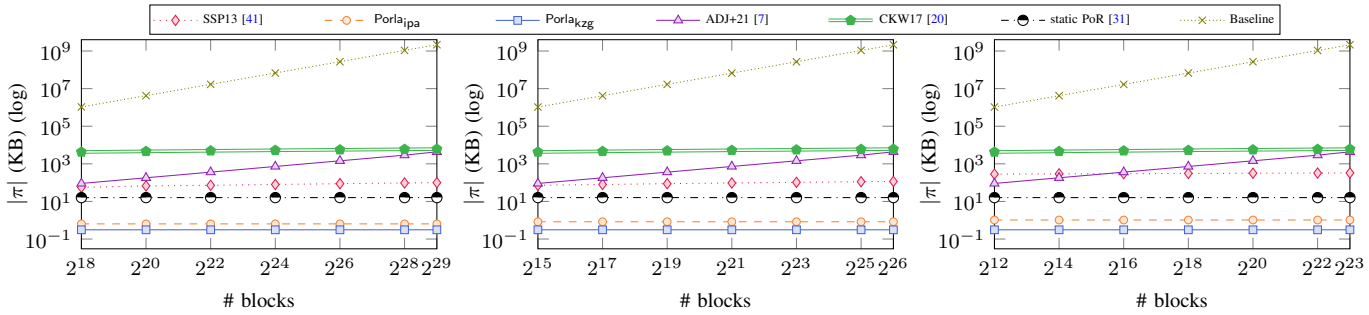
- Our schemes (Porla): We used `secp256k1` and `BN254` curves with 256-bit group order for $\mathsf{Porla}_{\mathsf{ipa}}$ and $\mathsf{Porla}_{\mathsf{kzg}}$, respectively. We selected a Proth prime $p = 207 \cdot 2^{248} + 1$ for the FFT codeword in both schemes.
- SSP13 scheme [41]: We used standard parameters suggested in [41]. Specifically, we used AES-GCM as the authenticated encryption scheme with a 128-bit key. We selected a Proth prime $p = 3 \cdot 2^{30} + 1$ for FFT codeword for fast modulo arithmetic. Note that the size of the FFT parameter does not affect the security of the system. We selected $\lambda = 128$ for the size of the secret matrix checksum.
- ADJ+21 scheme [7]: We ran their open-sourced implementation [29] and used standard parameters suggested in [7] for the performance benchmark. Specifically, we selected the 56-bit data chunk with $p = 144115188075855859$ (57-bit prime), matrix dimensions $m = n = \sqrt{N'/56}$, where $N' = N \cdot \beta/56$ is the total number of data chunks and SHA-512/224 for the Merkle hash algorithm. For the public audit setting, we used ristretto255, a 253-bit prime order subgroup of Curve25519, and $m = n = \sqrt{N'/252}$.
- Static PoR [31]: We embedded 10^6 128-bit sentinels into the database at random positions. We used $(255, 223, 32)$-Reed-Solomon Code as the erasure codes. We used AES to encrypt each 128-bit codeword. For each audit, the client verifies 1,000 random sentinels to detect if at least 1/2% of the database is corrupted as suggested in [31].
- CKW17 scheme [20]: We used $(255, 223, 32)$-Reed-Solomon code. For the underlying ORAM, we selected the expansion factor $\epsilon = 2$ so level i of the hierarchical ORAM has $3 \cdot 2^i$ slots to store 2^i blocks. The number of levels ranges from 26 to 37 for 1 GB to 2 TB database. We selected the stash size $|S| = 80$. For each audit, the client reads $\lambda = 128$ random data blocks with ORAM.
- Baseline: We used HMAC with SHA-256 to compute the checksum for each data block in the database.

C. Overall Results

Audit overhead. Figure 6 presents the proof size of our schemes compared with other schemes under different block sizes. We can see that our schemes incur minimal bandwidth overhead, in which our $\mathsf{Porla}_{\mathsf{kzg}}$ scheme only requires a constant bandwidth of 0.31 KB *regardless* of block size and database size, while $\mathsf{Porla}_{\mathsf{ipa}}$ requires only 0.64–1.03 KB for block sizes from 4 to 256 KB. Similar to $\mathsf{Porla}_{\mathsf{kzg}}$, the audit bandwidth overhead of $\mathsf{Porla}_{\mathsf{ipa}}$ is also independent of database size. Therefore, our schemes achieve an $87\times$–$1058\times$ smaller proof size than SSP13 scheme [41] which incurs 55.94–327.94 KB audit proof size. Note that the proof size of SSP13 scheme grows linearly with the block size and logarithmically with the database size, while our technique either incurs a constant size (i.e., $\mathsf{Porla}_{\mathsf{kzg}}$ scheme) or only grows logarithmically with the block size (i.e., $\mathsf{Porla}_{\mathsf{ipa}}$ scheme). We achieve *three to four* orders of magnitude smaller proof size than ADJ+21, which requires 91–4344 KB because its size is proportional to the square-root of the data size. Static PoR [31] has a constant proof size (16 KB) for all cases because the number of sentinels to check for each audit is fixed. Although CKW17 [20] reads a constant number of data blocks for each audit, it induces a much larger proof size (4264–6068 KB) than other schemes because it uses ORAM to access the blocks, which incurs a polylogarithmic bandwidth overhead. Since the baseline transmits the entire database, it incurs the largest proof size and bandwidth cost.

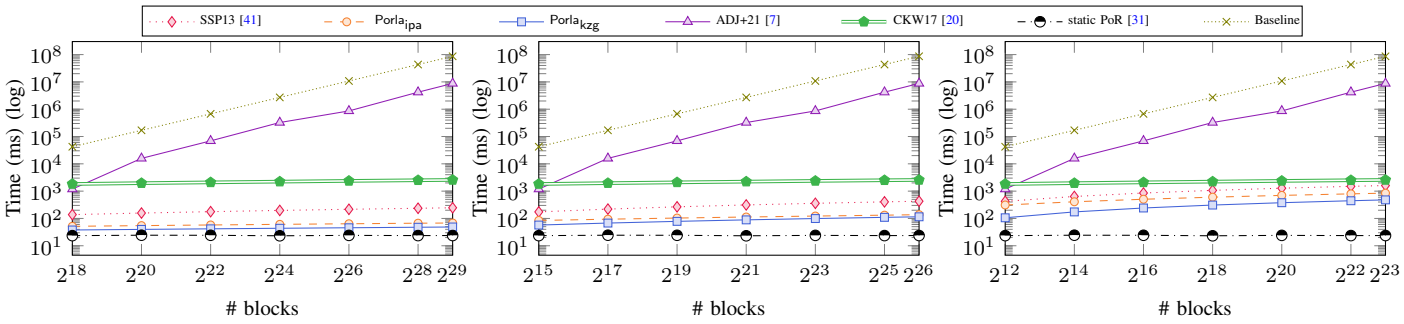
Figure 7 presents the (worst-case) end-to-end audit latency of our schemes compared with other works. $\mathsf{Porla}_{\mathsf{kzg}}$ achieves the lowest delay among all the schemes, where it is around $4\times$–$5\times$ faster than SSP13 scheme in all test cases, and up to $18000\times$–$180000\times$ faster than ADJ+21 scheme with 2 TB database, depending on the chosen block size. Another reason why Porla is much faster than ADJ+21 is that it incurs low disk I/O access (i.e., logarithmic to the database size). Specifically, Porla and SSP13 only read 7–13 MB (for 4 KB blocks), 43–52 MB (32 KB blocks), and 255–607 MB (256 KB blocks) of data from disk, while ADJ+21 requires reading the entire database, which incurs 1 GB–2 TB disk I/O access. The difference between $\mathsf{Porla}_{\mathsf{kzg}}$ and $\mathsf{Porla}_{\mathsf{ipa}}$ mainly stems from proving and verifying the polynomial evaluation. Static PoR [31] achieves the fastest audit time (≈ 23 ms) because the client only verifies a constant number of sentinels. However, we notice that this scheme does not support data update and only permits a limited number of audits. Other schemes (including ours) can perform an unlimited number of audits. CKW17 is slower than our schemes and static PoR since it uses ORAM to read the blocks, which costs 1.8–2.6 seconds. The baseline approach incurs the longest audit delay because it requires the entire database to be transmitted over the network.

Update overhead. We present the update latency of our schemes and their counterparts in Figure 8. As expected, our schemes incur a higher update delay than SSP13 and ADJ+21 schemes due to group operations. ADJ+21 scheme offers remarkable update performance since it only needs to update a Merkle path as well as the client secret vectors. Our



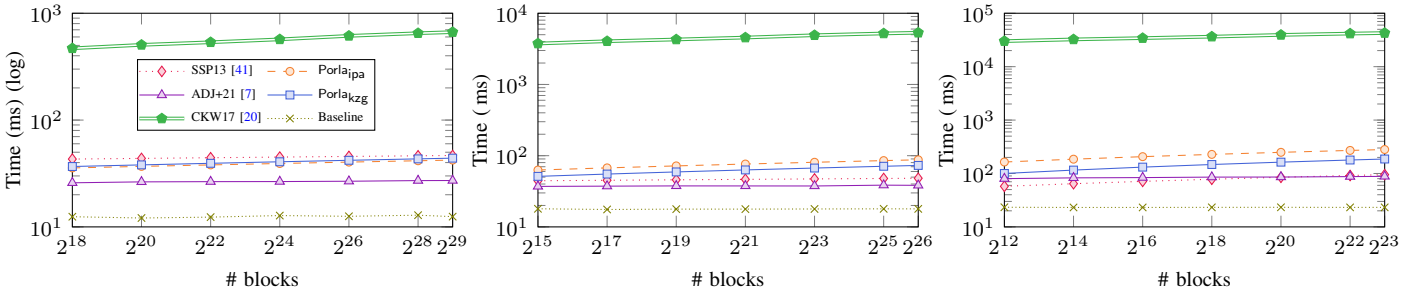
(a) $\beta = 4$ KB **(b)** $\beta = 32$ KB **(c)** $\beta = 256$ KB

Fig. 6: Audit proof size of Porla and its counterparts.



(a) $\beta = 4$ KB **(b)** $\beta = 32$ KB **(c)** $\beta = 256$ KB

Fig. 7: End-to-end audit delay of Porla and its counterparts.



(a) $|B| = 4$ KB **(b)** $|B| = 32$ KB **(c)** $|B| = 256$ KB

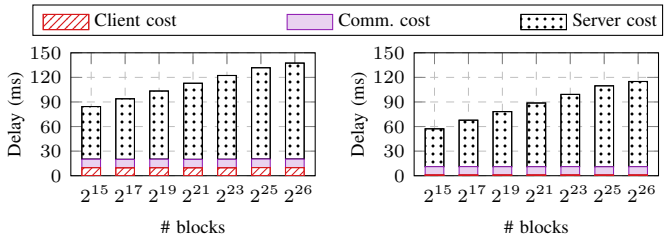
Fig. 8: End-to-end update delay of Porla and its counterparts.

schemes and SSP13 scheme update not only a Merkle path but also the erasure codewords and the MAC components. For 4 KB block size, our schemes are slightly faster than SSP13, where they take 36–44 ms to update the erasure codes and the MACs, while SSP13 takes 43–47 ms. For larger block sizes (i.e., 32 or 256 KB), our schemes are around 1.15×–3× slower than SSP13 since they incur group operations on the MACs, while SSP13 incurs symmetric operations. CKW17 supports dynamic update; however, it incurs a high update overhead (0.5–42.6 seconds) since it relies on ORAM to perform data write, which incurs polylogarithmic overhead as in the audit. The baseline scheme provides the most efficient update because the client simply sends the updated data block enclosed with its corresponding HMAC. By contrast, static PoR does not support the dynamic update feature.

We note that our schemes are more client-efficient than SSP13. Specifically, since the update token to update the MACs in our schemes can be precomputed, the client only needs to send the updated block and the updated Merkle path, and delegate all the computation to the server. In SSP13 scheme, the client has to (*i*) download the encrypted MACs from the server, (*ii*) decrypt them, (*iii*) compute the FFT erasure codes on the MACs, (*iv*) re-encrypt, and finally (*v*) upload them to the server along with the updated block and the Merkle path. We discuss the client cost of our schemes in more detail in §VII-D.

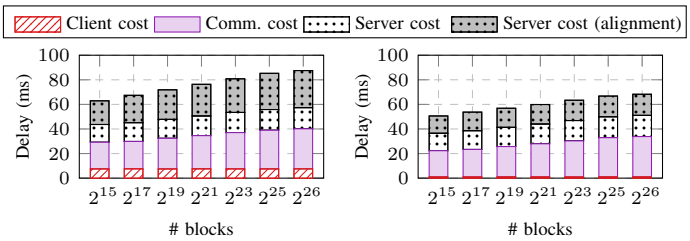
D. Detailed Cost Analysis

Figure 9 and Figure 10 present the detailed cost of data audit and update operations by our schemes, respectively, for 1 GB to 2 TB database with 32-KB block size. There are three factors that contribute to the total delay including client processing, communication latency, and server processing. We can see that our schemes incur a low processing cost at the client side in both audit and update operations, in which it only attributes 0.8%–11.6% to the total delay in audit, and 1.3%–11.9% in update. The client cost in Porla$_{ipa}$ scheme is higher than in Porla$_{kzg}$ for both data audit and update operations. This is mostly because the verification cost of polynomial evaluation via the inner product argument is higher than KZG during audit. Specifically, the verification time in Porla$_{kzg}$ is $\mathcal{O}(1)$, which takes less than 1 ms. In Porla$_{ipa}$, its cost is $\mathcal{O}(|B|)$,



(a) Porla$_{\text{ipa}}$ **(b)** Porla$_{\text{kzg}}$

Fig. 9: Detail audit cost in our schemes.



(a) Porla$_{\text{ipa}}$ **(b)** Porla$_{\text{kzg}}$

Fig. 10: Detail update cost in our schemes.

which takes 7.2–58.9 ms to verify for block sizes from 4 to 256 KB. In data update, since the client in Porla$_{\text{kzg}}$ scheme has access to the trapdoor τ that generates the common reference string, the cost to compute the commitment for the updated block is faster than that of Porla$_{\text{ipa}}$ scheme (i.e., curve multi-addition vs. multi-multiplication).

The network communication overhead during audit in our schemes is also minimal, which attributes 7.6%–17.7% to the total audit delay. This is due to the small audit proof size that our schemes offer as shown in Figure 6. In update, the network latency accounts for a large amount of the total delay due to the time to transmit the updated data block itself, as well as updated MAC hiding components for the rebuilt layer in $\mathbf{H}$. Specifically, it takes 21.4–32.9 ms (33.1%–45.0% of the total network delay) to update a 32-KB data block.

As shown in Figure 9 and Figure 10, a majority of overhead in our schemes stems from server processing, which attributes 53.5%–90.4% to the total delay. During an audit, the server incurs three main processing phases including (*i*) aggregating data blocks, (*ii*) aggregating corresponding MACs, and (*iii*) proving a polynomial evaluation at a random point. Aggregating data blocks and the MACs takes around 34 to 93 ms for 32- KB block size with the database of size from 1 GB to 2 TB. Since Porla$_{\text{kzg}}$ uses an efficient PC scheme, it only takes 0.8–13.9 ms to prove a block of sizes from 4 to 256 KB, compared with 4.2–134.9 ms in Porla$_{\text{ipa}}$ which accounts for 6.3%–43.4% of the total delay.

During a data update, most of the delay in server processing stems from the alignment computation discussed in §VII-A due to the unavailability of efficient standard curves that have a group order of a Proth prime. As shown in Figure 10 (the gray bar with dotted patterns), we can see that alignment processing costs 14.8–29.5 ms, which contributes 51.1%–63.8% to the total server cost and 28.9%–34.6% to the total delay. We expect that if such curves are found and implemented, the update overhead of our schemes can be further reduced by up to 34.6%, thereby significantly reducing the update gap in comparison with SSP13 and ADJ+21 schemes.

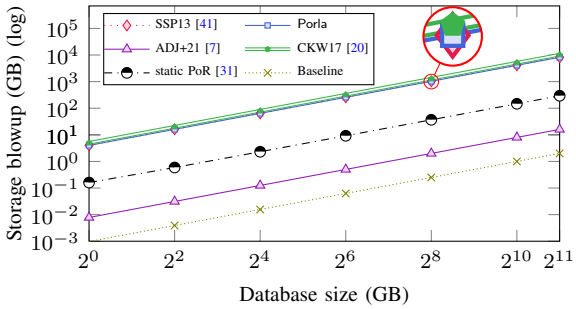


Fig. 11: Server storage blowup of (D)PoR schemes.

E. Storage

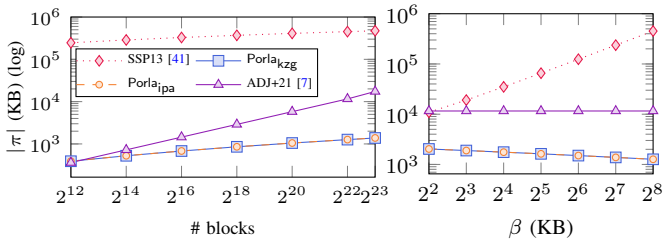
We present the storage overhead of our schemes at the client and server. Our schemes only require the client to maintain a 128-bit master key and a 128-bit counter, resulting in a total overhead of 32 bytes. This is far more efficient than other works. For instance, SSP13 scheme requires the client to store a 128-bit master key, a 128-bit counter, and an authentication matrix of size $\mathbb{F}_p^{\lambda/\beta_0 \times \beta/\beta_0}$, where $\beta_0 = \log_2 p$ (i.e., 64 KB (resp. 512 KB) for 32 KB (resp. 256 KB) block size and 128-bit security).

With a database of size $N\beta$ bits, the extra server storage of our schemes is $5N\beta + N|\mathsf{H}| + 5N|\mathbb{G}|$ bits due to the erasure codes, the Merkle tree for the read buffer, and the MAC components, respectively. SSP13 incurs similar server storage overhead as our schemes, except that the size of its MAC components is two times smaller than ours because they are symmetric block ciphers rather than group elements. ADJ+21 scheme achieves more lightweight server storage, which incurs extra storage of $(2N-1)|\mathsf{H}|$ bits because it does not require erasure codes. The storage overhead of the static PoR [31] is 14.4%–16.1% since it encodes the database with erasure codes and embeds the sentinels at random positions. The baseline brings the lowest price regarding extra storage overhead because it stores the original data blocks in which each data block is enclosed with a 256-bit HMAC. Finally, CKW17 requires approximately $2(1+\epsilon)\gamma N\beta + 80\beta$ bits of server storage, where $\epsilon = 2$ and $\gamma = 255/223$ are the expansion factors of the hierarchical ORAM and Reed-Solomon codes, respectively.

Figure 11 presents the extra server storage overhead of our schemes and their counterparts when storing different sizes of the database (from 1 GB to 2 TB) under a 32 KB block size. Note that both Porla$_{\text{kzg}}$ and Porla$_{\text{ipa}}$ instantiations incur the same extra server storage overhead and their cost is nearly similar to SSP13 scheme (i.e., only 0.1% difference between them). Compared with CKW17, our schemes incur 20% less server storage blowup.

F. Public Audit Experiments

We assess the performance of Porla in the public audit setting. Figure 12 presents the proof size of the public audit of Porla compared with other works. As shown in Figure 12a, the proof size of Porla is two orders of magnitude smaller than SSP13 scheme, where it only costs 391–1384 KB, compared with 246120–472302 KB. Our proof size also grows more slowly than ADJ+21 since it is only logarithmically proportional to the database size, compared with square-root in ADJ+21. Concretely, our proof size is 12× smaller than



(a) $\beta = 256$ KB **(b)** $|\mathsf{DB}| = 1$ TB

Fig. 12: Proof size of public audit of Porla and its counterparts.

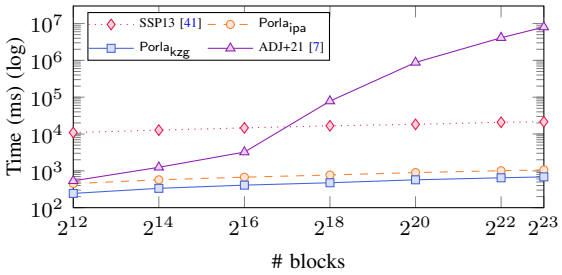


Fig. 13: Delay of public audit of Porla and its counterparts.

ADJ+21 for a 2 TB database.

Figure 12b presents the impact of data block size on the public audit proof size. Unlike SSP13 scheme, our audit proof size is independent of block sizes. In fact, we can see that our proof size decreases when the block size increases (with fixed database size). This is because when the database size is fixed, increasing the block size reduces the number of hierarchical levels in the erasure codes, thereby reducing the number of random positions being checked. Since the proof size in ADJ+21 only grows square-root to the database, we can see that it is constant regardless of block size. However, its public variant uses group operations, and thus the proof size in that case is $8\times$ larger than its private audit setting.

Figure 13 presents the public audit latency of Porla compared with its counterparts under 256- KB block size. We can see that the public audit of Porla only incurs a small extra overhead over its private audit version (i.e., 140–215 ms), while SSP13 observes a significant increase (i.e., $20\times$) due to its increased proof size. For ADJ+21 scheme, its public audit incurs an extra delay of 0.6–763.4 seconds due to group operations performed by the client. In general, our public audit is $20\times$–$45\times$ faster than SSP13, and three orders of magnitude faster than ADJ+21.

Acknowledgment

Attila A. Yavuz is supported by an unrestricted gift from Cisco Research Award and the NSF CAREER Award (CNS-1917627). Elaine Shi is supported by NSF grants 2128519 and 2044679, a DARPA SIEVE grant, and a Packard Fellowship. Thang Hoang is supported by an unrestricted gift from Robert Bosch, and the Commonwealth Cyber Initiative (CCI), an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit www.cyberinitiative.org.

References

[1] Amazon s3 glacier storage classes. https://aws.amazon.com/s3/storage-classes/glacier/.

[2] Audit compliance: Definition and what it means for the cloud. https://www.ionos.com/digitalguide/online-marketing/online-sales/audit-compliance/.

[3] Azure archive storage. https://azure.microsoft.com/en-us/services/storage/archive/#overview.

[4] Zeromq: An open-source universal messaging library.

[5] Aydin Abadi, Steven J Murdoch, and Thomas Zacharias. Recurring contingent payment for proofs of retrievability. *Cryptology ePrint Archive*, 2021.

[6] Shweta Agrawal and Dan Boneh. Homomorphic macs: Mac-based integrity for network coding. In *International conference on applied cryptography and network security*, pages 292–305. Springer, 2009.

[7] Gaspard Anthoine, Jean-Guillaume Dumas, Mélanie de Jonghe, Aude Maignan, Clément Pernet, Michael Hanling, and Daniel S Roche. Dynamic proofs of retrievability with low server storage. In *30th USENIX Security Symposium*, pages 537–554, 2021.

[8] Frederik Armknecht, Jens-Matthias Bohli, Ghassan Karame, and Wenting Li. Outsourcing proofs of retrievability. *IEEE Transactions on Cloud Computing*, 9(1):286–301, 2018.

[9] Frederik Armknecht, Jens-Matthias Bohli, Ghassan O Karame, Zongren Liu, and Christian A Reuter. Outsourced proofs of retrievability. In *ACM SIGSAC CCS'2014*, pages 831–843, 2014.

[10] Amit Ashbel. The complete guide to cold data storage. https://cloud.netapp.com/blog/cvo-blg-the-complete-guide-to-cold-data-storage.

[11] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609, 2007.

[12] Giuseppe Ateniese, Roberto Di Pietro, Luigi V Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, pages 1–10, 2008.

[13] Microsoft Azure. Azure blob storage documentation. https://docs.microsoft.com/en-us/azure/storage/blobs/.

[14] Microsoft Azure. Hot, cool, and archive access tiers for blob data. https://docs.microsoft.com/en-us/azure/storage/blobs/access-tiers-overview.

[15] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *Annual Cryptology Conference*, pages 111–131. Springer, 2011.

[16] Gautam Botrel, Thomas Piellard, Youssef El Housni, Arya Tabaie, and Ivo Kubjas. Consensys/gnark-crypto: v0.6.1, February 2022.

[17] Kevin D Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: Theory and implementation. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 43–54, 2009.

[18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[19] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 229–243, 2017.

[20] David Cash, Alptekin Küpçü, and Daniel Wichs. Dynamic proofs of retrievability via oblivious ram. *Journal of Cryptology*, 2017.

[21] Dario Catalano and Dario Fiore. Practical homomorphic macs for arithmetic circuits. In *EUROCRYPT'13*, pages 336–352. Springer, 2013.

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

[23] Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. Mr-pdp: Multiple-replica provable data possession. In *ICDCS'08*, pages 411–420. IEEE, 2008.

[24] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In *Annual Cryptology Conference*, pages 585–605. Springer, 2015.

[25] C Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. *ACM Transactions on Information and System Security (TISSEC)*, 17(4):1–29, 2015.

[26] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, pages 186–194. Springer, 1986.

[27] Ben Fisch. Poreps: Proofs of space on useful data. *Cryptology ePrint Archive*, 2018.

[28] Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. Pointproofs: Aggregating proofs for multiple vector commitments. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 2007–2023, 2020.

[29] Michael Hanling and Daniel Roche. Linear algebra-based proof of retrievability protocol for ensuring data integrity. https://github.com/dsroche/la-por.

[30] Intel. What is storage as a service? https://www.intel.com/content/www/us/en/cloud-computing/storage-as-a-service.html.

[31] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597, 2007.

[32] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *International conference on the theory and application of cryptology and information security*, pages 177–194. Springer, 2010.

[33] Karen Ann Kent and Murugiah Souppaya. Guide to computer security log management:. 2006.

[34] Julien Lavauzelle and Françoise Levy-dit Vehel. New proofs of retrievability using locally decodable codes. In *2016 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2016.

[35] RD McDowall. *Data integrity and data governance: practical implementation in regulated laboratories*. Royal Society of Chemistry, 2018.

[36] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[37] Nicholas Pippenger. On the evaluation of powers and monomials. *SIAM Journal on Computing*, 9(2):230–250, 1980.

[38] Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In *Theory of Cryptography Conference*, pages 262–285. Springer, 2016.

[39] Ron Ross, Victoria Pillitteri, Kelley Dempsey, Mark Riddle, and Gary Guissanie. Protecting controlled unclassified information in nonfederal systems and organizations. Technical report, National Institute of Standards and Technology, 2019.

[40] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *International conference on the theory and application of cryptology and information security*, pages 90–107. Springer, 2008.

[41] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In *ACM CCS'13*, pages 325–336, 2013.

[42] Victor Shoup et al. Ntl: A library for doing number theory, 2001.

[43] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: A scalable cloud file system with efficient integrity checks. In *ACSAC'12*, pages 229–238, 2012.

[44] Dimitrios Vasilopoulos, Kaoutar Elkhiyaoui, Refik Molva, and Melek Onen. Poros: proof of data reliability for outsourced storage. In *Proceedings of the 6th International Workshop on Security in Cloud Computing*, pages 27–37, 2018.

[45] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555, 2017.

[46] Huaqun Wang. Proxy provable data possession in public clouds. *IEEE Transactions on Services Computing*, 6(4):551–559, 2012.

[47] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *European symposium on research in computer security*, pages 355–370. Springer, 2009.

[48] Pieter Wuille. libsecp256k1. https://github.com/bitcoin-core/secp256k1.

[49] Yan Zhu, Huaixi Wang, Zexing Hu, Gail-Joon Ahn, Hongxin Hu, and Stephen S Yau. Efficient provable data possession for hybrid clouds. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 756–758, 2010.

APPENDIX

A. *Proof of Theorem 1*

We show that our MAC achieves unforgeability via a sequence of indistinguishable hybrid games G_0, G_1. Let W_0 and W_1 be the events that $\mathcal{A}$ wins at G_0 and G_1, respectively.

Game G_0. This is the game presented in Definition 6 applied to our Σ scheme in Figure 2. So, $\Pr[W_0] = \mathsf{Adv}[\mathcal{A}, \Sigma]$.

Game G_1. G_1 is identical to G_0, except that we replace the PRF function F in our scheme with a truly random function. Specifically, $r \leftarrow F(k, st)$ is replaced with $r \xleftarrow{\$} \mathbb{Z}_p$. Let $\mathcal{A}_{\mathsf{prf}}$ be the PRF adversary against F. We have that

$$|\Pr[W_0] - \Pr[W_1]| = \mathsf{Adv}[\mathcal{A}_{\mathsf{prf}}, F]$$

We now give the detail of how the challenger $\mathcal{C}$ in G_1 proceeds as follows.

Init. $\alpha \xleftarrow{\$} \mathbb{Z}_p$, $\mathcal{L} := \{\emptyset\}$,

Query. At each time step, $\mathcal{A}$ submits to $\mathcal{C}$ either a signing query or an update query as follows.

- **Signing Query.** $\mathcal{A}$ submits a signing query $q = \left(\mathbf{g}^{\mathbf{v}}, st\right)$. If $st \in \mathcal{L}$, reject. Otherwise, sample $r \xleftarrow{\$} \mathbb{Z}_p$, computes $\sigma \leftarrow \mathbf{g}^{\mathbf{v} \cdot \alpha} h^r$ and $\mathcal{L} \leftarrow \mathcal{L} \cup (st; \mathbf{g}^{\mathbf{v}}, r)$ and send σ to $\mathcal{A}$.
- **Update Query.** $\mathcal{A}$ submits an update query $((st'_1, c_1), \ldots, (st'_m, c_m), st)$. If $(st \in \mathcal{L}) \wedge \exists st'_j \notin \mathcal{L}$ for some $j \in [m]$, reject. Otherwise, get $(\mathbf{g}^{\mathbf{v}_j}, r'_j) \leftarrow \mathcal{L}(st_j)$ for $j \in [m]$, sample $r \xleftarrow{\$} \mathbb{Z}_p$, compute $\tilde{r} \leftarrow r - \sum_{j=1}^{m} c_j r'_j$, $\tau \leftarrow h^{\tilde{r}}$, $\mathbf{g}^{\tilde{\mathbf{v}}} \leftarrow \prod_{j=1}^{m} \mathbf{g}^{\mathbf{v}_j \cdot c_j}$, $\mathcal{L} \leftarrow \mathcal{L} \cup (st; \mathbf{g}^{\tilde{\mathbf{v}}}, \tilde{r})$ and send τ to $\mathcal{A}$.

Output. $\mathcal{A}$ outputs $\left(\mathbf{g}^{\mathbf{v}^*}, \sigma^*, (st_1^*, c_1^*), \ldots, (st_m^*, c_m^*)\right)$. To determine if $\mathcal{A}$ wins the game, we first analyze $\mathcal{A}$'s transcript $\mathcal{T}$ when querying signing and update oracles.

Transcript. $\mathcal{T}$ contains a list of tags and update tokens, each of which corresponds to a unique state. Specifically, $\mathcal{T} := \{(st_1, t_1), (st_2, t_2), \ldots, (st_n, t_n)\}$ where $t_i \in \{\sigma_i, \tau_i\}$. We can see that each t_i contains an independent random $r_i \in \mathbb{Z}_p$. Thus, the tags and tokens from $\mathcal{A}$'s view are independent and indistinguishable. Thus, the probability that $\mathcal{A}$ can determine α and randoms being used to create the tags and tokens from the protocol transcript is $1/p$.

Next we consider the following cases:

- If $\exists st_i^* \notin \mathcal{L}$ for some $i \in [m]$, set $r_i^* \xleftarrow{\$} \mathbb{Z}_p$. Set $r_j^* \leftarrow r_j$ for all other $st_j^* \in \mathcal{L}$, where $r_j \leftarrow \mathcal{L}(st_j)$ (type-1 forgery).
- Else, set $r_i^* \leftarrow r_i$, where $r_i \leftarrow \mathcal{L}(st_i^*)$ for all $i \in [m]$

$\mathcal{A}$ wins (i.e., event W_1 happens) if

$$\begin{cases} \sigma^* = (\mathbf{g}^{\mathbf{v}^*}_m)^{\alpha} h^{\sum_{i=1}^{m} c_i^* \cdot r_i^*}, \text{ and} & (5) \\ \mathbf{g}^{\mathbf{v}^*} \neq \prod_{i=1}^{m} \mathbf{g}^{\mathbf{v}_i \cdot c_i^*}, \text{ where } \mathbf{g}^{\mathbf{v}_i} \leftarrow \mathcal{L}(st_i^*) \forall i \in [m], \text{ and} & (6) \\ c_i^* \neq 0 \ \forall i \in [m]. & (7) \end{cases}$$

Let T and $\neg T$ be the event that $\mathcal{A}$ outputs a type-1 forgery and a type-2 forgery, respectively.

Type-1 forgery. Recall that $\mathbf{g} = (g_1, \ldots, g_n)$ are public generators. Let $\hat{g}$ be a subgroup generator $\hat{g}$ such that $\hat{g}^{a_i} = g_i$

for $i \in [n]$ with some arbitrary a_i, and $\hat{g}^x = h$ for some x. Let $\mathbf{v}^* = (v_1^*, \ldots, v_n^*)$ be some vector committed to $\mathbf{g}^{\mathbf{v}^*}$. (5) can be rewritten as $\hat{g}^{\mu} = \hat{g}^{\sum_{i=1}^{n} \alpha(a_i v_i^*) + \sum_{i=1}^{m} x(c_i^* r_i^*)}$. For this to hold, $\mathcal{A}$ needs to find $\mu \in \mathbb{Z}_p$ such that

$$\begin{aligned} \mu &= \sum_{i=1}^{n} \alpha(a_i v_i^*) + \sum_{i=1}^{m} x(c_i^* r_i^*) \\ &= \alpha A + xB \end{aligned} \tag{8}$$

where $A = \sum_{j=1}^{n} a_i v_i^*$ and $B = \sum_{i=1}^{m} c_i^* r_i^*$.

Assuming even DLP is solved, i.e., $\mathcal{A}$ knows all a_i for $i \in [n]$, x and thus, can determine $\mathbf{v}^*$ from $\mathbf{g}^{\mathbf{v}^*}$ and compute A. However, there always exists at least an unknown random r_i^* in B. Thus, the right-hand side of (8) is independent of $\mathcal{A}$'s view and indistinguishable from a random value in $\mathbb{Z}_p$ due to α and B. Therefore, when T happens, the probability that (8) holds is $1/p$. Thus $\Pr[W_2 \wedge T] = (1/p) \cdot \Pr[T]$.

Type-2 forgery. $\mathcal{A}$ outputs $(st_1^*, \ldots, st_m^*)$, *all* of which were previously queried in the query phase. Let $(\mathbf{g}^{\mathbf{v}_i}, r_i) \leftarrow \mathcal{L}(st_i^*)$ for $i \in [m]$. $\mathcal{A}$ wins if $\mathbf{g}^{\mathbf{v}^*} \neq \prod_{i=1}^{m} \mathbf{g}^{\mathbf{v}_i \cdot c_i}$ and (5) holds.

Let $\sigma_1, \ldots, \sigma_m$ be the tags of $\mathbf{g}^{\mathbf{v}_1}, \ldots, \mathbf{g}^{\mathbf{v}_m}$ under state $st_1^*, \ldots, st_m^*$, respectively, i.e., $\sigma_i = \mathbf{g}^{\mathbf{v}_i \cdot \alpha} h^{r_i}$ for $i \in [m]$. Define $\mathbf{g}^{\mathbf{v}'} := \prod_{i=1}^{m} \mathbf{g}^{\mathbf{v}_j \cdot c_i^*}$ and $\sigma' := \prod_{i=1}^{m} \sigma_i{}^{c_i^*}$ It is easy to see that σ' is a valid tag for $\mathbf{g}^{\mathbf{v}'}$. So, the following two relations hold: $\mathbf{g}^{\mathbf{v}^* \cdot \alpha} \prod_{i=1}^{m} h^{c_i^* \cdot r_i^*} = \sigma$ and $\mathbf{g}^{\mathbf{v}' \cdot \alpha} \prod_{i=1}^{m} h^{c_i^* \cdot r_i} = \sigma'$ Since $r_i = r_i^* \forall i \in [m]$, this is equivalent to $\mathbf{g}^{\alpha \cdot (\mathbf{v}^* - \mathbf{v}')} = \sigma / \sigma'$

To produce a valid type-2 forgery, $\mathcal{A}$ needs to find a pair $(\mathbf{g}^{\mathbf{v}^*}, \sigma)$ satisfying the above relation. We have that $\mathbf{g}^{\mathbf{v}^*} \neq \prod_{i=1}^{m} \mathbf{g}^{\mathbf{v}_j \cdot c_j^*}$ (by (6)), $\mathbf{v}^* \neq \sum_{j=1}^{m} c_j^* \cdot \mathbf{v}_j$, and thus $\mathbf{v}^* \neq \mathbf{v}'$.

Since $\alpha \neq 0$ is indistinguishable from a random in $\mathbb{Z}_p$ in the $\mathcal{A}$'s view (as proven in the protocol transcript analysis above), the probability that $\mathcal{A}$ can find $\mathbf{v}^*$ resulting in $\mathbf{g}^{\mathbf{v}^*}$ and σ satisfying the relation is $1/p$.

So we have that $\Pr[W_2 \wedge \neg T] = (1/p) \cdot \Pr[\neg T]$. Putting it all together, we have that

$$\Pr[W_2] = \Pr[W_2 \wedge T] + \Pr[W_2 \wedge \neg T] = \frac{1}{p}(\Pr[T] + \Pr[\neg T]) = \frac{1}{p}.$$

B. Proof of Theorem 2

In the data read protocol, the authenticity is straightforward due to soundness and completeness of the underlying VC scheme. In audit protocol, suppose $(\mathbf{v}_1, \ldots, \mathbf{v}_t)$ are data blocks being audited. Let $(\mathbf{g}^{\mathbf{v}_1}, \ldots, \mathbf{g}^{\mathbf{v}_t})$ be the commitments of $(\mathbf{v}_1, \ldots, \mathbf{v}_t)$. In the first step, the client requests a pair $(\prod_{i=1}^{m} \mathbf{g}^{\mathbf{v}_i \cdot \rho^i}, \prod_{i=1}^{m} \sigma_i^{\rho^i})$, where $(\sigma_1, \ldots, \sigma_t)$ are the tags of $(\mathbf{g}^{\mathbf{v}_1}, \ldots, \mathbf{g}^{\mathbf{v}_t})$ created under specific states $(st_1, \ldots, st_t)$, respectively, and ρ is a random scalar. Due to the unforgeability of our MAC scheme, each $(st_i, \mathbf{g}^{\mathbf{v}_i}, \sigma_i)$ is well-formed, i.e., $\sigma_i = \mathbf{g}^{\mathbf{v}_i \alpha} h^{F(k, st_i)}$ for each $i \in [t]$. That means $(\prod_{i=1}^{t} \mathbf{g}^{\mathbf{v}_i \rho^i}, \prod_{i=1}^{\lambda} \sigma_i{}^{\rho^i})$ is well-defined by $(st_1, \ldots, st_t)$ and ρ indicated by the client. By Theorem 1, the probability that the malicious server can cheat the client to accept $(\prod_{i=1}^{m} \mathbf{g}^{\mathbf{v}_i' \cdot \rho^i}, \prod_{i=1}^{m} \sigma_i'{}^{\rho^i})$, where $\mathbf{g}^{\mathbf{v}_i'} \neq \mathbf{g}^{\mathbf{v}_i}$ or $\sigma_i' \neq \sigma_i$ for some i is $\mathsf{negl}(\lambda)$. In other words, the adversarial server must correctly output $(\prod_{i=1}^{\lambda} \mathbf{g}^{\mathbf{v}_i \cdot \rho^i}, \prod_{i=1}^{\lambda} \sigma_i{}^{\rho^i})$, otherwise the client will reject with overwhelming probability.

In the final step of the audit, the client requests the server to evaluate the polynomial $f(X) = \sum_{d=1}^{D} a_d \cdot X^{d-1}$, where $a_d = \sum_{j=1}^{t} \rho^j \cdot \mathbf{v}_j[d]$, at a random point α and verify the evaluation. As we make use of PC schemes (i.e., KZG and IP-based) with completeness and soundness properties, the probability that the malicious server can deviate from the protocol and fool the client to accept a wrong proof/evaluation is $\mathsf{negl}(\lambda)$.

All the above arguments indicate that the malicious server must always follow the protocol faithfully as the honest server, otherwise, the client will reject with overwhelming probability, and this completes the authenticity proof of our schemes.

C. Proof of Theorem 3

Let $P = (\mathsf{op}_1, \ldots, \mathsf{op}_q)$, where $\mathsf{op}_i \in \{\mathsf{PRead}(j), \mathsf{PWrite}(j, \mathbf{b}'), \mathsf{PAudit}\}$ for some $j \in [N]$ be the sequence of interaction between the client $\mathcal{C}$ and malicious server $\mathcal{S}^*$. Let $\mathcal{C}_{\mathsf{fin}}, \mathcal{S}_{\mathsf{fin}}^*$ be their final state after the interaction.

We start by proving the following lemma.

Lemma 2. *There exists an extractor $\hat{\mathcal{E}}$ that, given the access to the malicious server execution during* PAudit *protocol, it can extract audited blocks with overwhelming probability.*

Proof: Remark that each of our PAudit protocol execution invokes four main steps as follows.

1) The client specifies t random blocks $(\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_t})$ with corresponding indices $(i_1, \ldots, i_t)$ to be audited.
2) The client sends a random scalar ρ and requests the server to perform a random linear combination of t audit blocks based on ρ as $\sum_{j=1}^{t} \mathbf{v}_{i_j} \cdot \rho^{i_j}$.
3) The server returns the commitment of polynomial representing the aggregated block by random linear combination as $\mathbf{g}^{\mathbf{v}}$, where $\mathbf{v} = \sum_{j=1}^{t} \mathbf{v}_{i_j} \cdot \rho^{i_j}$.
4) The client sends random evaluation point α and requests the server to evaluate the committed polynomial at α, and verifies the proof.

Let $\hat{\mathcal{C}}$ and $\hat{\mathcal{S}}^*$ be the client state and the server state after step 1. Let $\tilde{\mathcal{C}}$ and $\tilde{\mathcal{S}}^*$ be the client state and the server state after step 3. Let $\mathbf{v} \leftarrow \mathcal{E}'^{\tilde{\mathcal{S}}^*}(\tilde{\mathcal{C}}, 1^\lambda)$ be the extractor of the underlying PC scheme with knowledge soundness that can extract the polynomial $f(X) = \sum_{d=0}^{D} \mathbf{v}[d+1] \cdot X^d$.

The Extractor. We define an extractor $\hat{\mathcal{E}}^{\hat{\mathcal{S}}}(\hat{\mathcal{C}}, 1^t, 1^\lambda)$ that can extract all individual $\mathbf{v}_{i_j}$ for $j \in [t]$ using the extractor of polynomial commitment as the subroutine as follows.

1) Initialize an empty key-value table $\mathbf{F} := \{\emptyset\}$.
2) Keep rewinding and executing the following step in $n = \max(2t, \lambda) \cdot m = \mathsf{poly}(\lambda)$ times:
   a) Pick a random scalar $\rho \in \mathbb{F}_p$, and continue the remaining steps of the audit with $\mathcal{S}^*$. If step 3 is accepted, invoke $\mathcal{E}'^{\tilde{\mathcal{S}}}(\tilde{\mathcal{C}}, 1^\lambda)$ in step 4 to extract $f(X)$ and set $\mathbf{F}(\rho) := f(X)$. Rewind $\mathcal{C}$ and $\mathcal{S}^*$ to the state prior to this execution (i.e., $\hat{\mathcal{C}}$ and $\hat{\mathcal{S}}$) and continue the next round.
   b) If the size of $\mathbf{F}$ is $|\mathbf{F}| \leq t$, output fail. Otherwise, pick t distinct pairs in $\mathbf{F}$ as $(\rho_i, f_i(X))$ for $i \in [t]$. Let $\mathbf{V} = \mathsf{vand}[\rho_1, \ldots, \rho_t]$ and $\mathbf{V}^{-1}$ be a Vandermonde matrix and its inverse, respectively, $\mathbf{f} = (f_1(X), \ldots, f_t(X))$. Compute $\mathbf{v}_{i_j} = \mathbf{V}^{-1}[j, *] \cdot \mathbf{f}$ for $j \in [t]$.

We argue that the extractor $\hat{\mathcal{E}}$ must either output all correctly aggregated polynomials in $\mathbf{F}$ or return fail. In other words, the

extractor will never put an incorrectly aggregated polynomial into $\mathbf{F}$ and always detect failure. This is achieved by the authenticity property of our scheme by Definition 3, specifically the unforgeability of our MAC scheme for polynomial (homomorphic) commitment and the knowledge soundness of underlying PC scheme, both of which guarantee that $\hat{\mathcal{E}}$ never extracts and outputs incorrectly aggregated polynomial into $\mathbf{F}$. Specifically, let $\hat{p}$ be the probability of the bad event, where one of the executions by $\hat{\mathcal{E}}$ output some incorrectly aggregated polynomial into $\mathbf{F}$ without being rejected by the client. In other words, the bad event happens if the malicious server can break either the unforgeability of our MAC scheme or the knowledge soundness of underlying PC scheme. Since the extractor executes steps 2-4 in PAudit protocol with rewinding in $n = \mathsf{poly}(\lambda)$ times, there is at least $\hat{p}/n$ probability that the bad event can happen in a single random execution with $\hat{\mathcal{S}}^*$. However, that also means $\hat{\mathcal{S}}^*$ can be used to break either the unforgeability of our MAC scheme or the knowledge soundness of PC scheme with advantage $\hat{p}/\mathsf{poly}(\lambda)$. By Definition 3, we must have that $\hat{p} = \mathsf{negl}(\lambda)$.

We now show that $\hat{\mathcal{E}}$ will never return fail. Let E be the event that fail is returned. For each round $i \in [n]$ executed by the extractor in step 2, we introduce random variables as

1) Let $X_i \in \{0,1\}$ be an indicator random variable, where $X_i = 1$ if the execution at round i does not reject.
2) Let $G_i = \{\rho \in \mathbf{F}\}$ be a random variable indicating the set of keys in $\mathbf{F}$ at the beginning of round i.
3) Let $Y_i \in \{0,1\}$ be an indicator random variable, where $Y_i = 1$ if $|G_i| < t$ and the random scalar ρ that the extractor chooses at round i is already used in some previous rounds, i.e., $\rho \in G_i$.

At the iteration round i, if $(X_i = 1 \wedge Y_i = 0)$, there is a new pair of $(\rho, f(X))$ being added to $\mathbf{F}$, i.e., $|G_{i+1}| = |G_i|+1$ and, therefore, after $n = \mathsf{poly}(\lambda)$ iterations, $|G_n| \geq t$. That means the event E only happens if $\sum_{i=1}^{n} X_i \leq t$ or there exists a round i where $Y_i = 1$. So we have that

$$\Pr[E] \leq \Pr\left[\sum_{i=1}^{n} X_i < t\right] + \sum_{i=1}^{n} \Pr\left[Y_i = 1\right]$$

We have that $\sum_{i=1}^{n} \Pr\left[Y_i = 1\right] \leq \frac{t(t-1)}{|\mathbb{F}_p|}$. Let $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$, we have that

$$\begin{aligned}\Pr\left[\sum_{i=1}^{n} X_i < t\right] &\leq \Pr\left[\bar{X} < 1/p' - \left(1/p' - \frac{t}{n}\right)\right] \\ &\leq \exp\left(-2n\left(1/p' - t/n\right)^2\right) \leq \exp(-n/p') \\ &\leq 2^{-\lambda}\end{aligned}$$

where the second and third inequalities are due to Chernoff-Hoeffding bound and $(p' = \mathsf{poly}(\lambda) \wedge 1/p' < 1/2)$, respectively. We have that $\Pr[E] \leq \frac{t(t-1)}{|\mathbb{F}_p|} + 2^{-\lambda} = \mathsf{negl}(\lambda)$. This means the extractor can always extract correct audited blocks in each audit protocol execution. ■

We now prove that our PAudit protocol achieves retrievability by constructing another extractor $\mathcal{E}$ that can recover the entire database content using the internal $\mathcal{E}^*$ within the PAudit protocol as the subroutine.

The Extractor. We define the extractor $\mathcal{E}^{\mathcal{S}^*_{\mathsf{fin}}}(\mathcal{C}_{\mathsf{fin}}, 1^N, 1^\lambda)$ as:

1) Init an empty database $\mathbf{M} := (\perp)^{N'}$, where $N' = 2N$.
2) Keep rewinding and auditing the server by repeating the following step for $n = \max(2N, \lambda) \cdot c = \mathsf{poly}(\lambda)$ times:
   a) Execute protocol Audit with $\mathcal{S}^*$ by picking t distinct random indices $(i_1, \ldots, i_t)$ at step 1 of the protocol. Invoke the extractor $\hat{\mathcal{E}}$ at step 2 to extract audited blocks as $(\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_t}) \leftarrow \hat{\mathcal{E}}^{\hat{\mathcal{S}}^*_{\mathsf{fin}}}(\hat{\mathcal{C}}^*_{\mathsf{fin}}, 1^t)$. If the protocol is accepting, set $\mathbf{M}[i_1] = \mathbf{v}_1, \ldots, \mathbf{M}[i_t] = \mathbf{v}_{i_t}$. Rewind $\mathcal{C}$ and $\mathcal{S}^*$ to the state prior to this execution (i.e., $\mathcal{C}_{\mathsf{fin}}$ and $\mathcal{S}^*_{\mathsf{fin}}$) and continue the next iteration round.
   b) If the number of "filled in" values in $\mathbf{M}$ is $|\{j \in [N'] : \mathbf{M}[j] \neq \perp\}| < \delta \cdot N'$, where $\delta = \frac{1}{2}$ then output fail. Otherwise, by Lemma 1, apply corresponding erasure decoding to $\mathbf{M}$ to recover and output $(\mathbf{b}_1, \ldots, \mathbf{b}_N)$.

We can now simply follow the proof in [20] to show that the extractor can always recover the entire database. For completeness, we present the main arguments here. We first argue that the extractor must either output the correct database content $\mathbf{M}$ or return fail. In other words, the extractor will never output an incorrect database $\mathbf{M}$ and always detect failure. This is achieved by the authenticity property in Definition 3 of our PAudit protocol and specifically, the extractability property by Lemma 2, which guarantees that $\mathcal{E}$ never extracts and outputs incorrect blocks into $\mathbf{M}$ per audit iteration.

We show that the extractor never returns fail. Let E be the event, where fail is returned. For each round $i \in [n]$ executed by $\mathcal{E}$ at step 2, we introduce random variables as follows.

1) Let $X_i \in \{0,1\}$ be an indicator random variable, where $X_i = 1$ if the audit at round i does not reject.
2) Let $G_i = \{j \in [N'] : \mathbf{M}[j] \neq \perp\}$ be a random variable indicating the subset of filled-in values in $\mathbf{M}$ at the beginning of round i.
3) Let $Y_i \in \{0,1\}$ be an indicator random variable, where $Y_i = 1$ if $|G_i| < \delta \cdot N'$ and *all* the positions $(j_1, \ldots, j_t)$ that $\mathcal{E}^{\mathcal{S}^*}$ chooses to audit at round i are already checked in previous rounds, i.e., $(j_1, \ldots, j_t) \in G_i$.

At the iteration round i, if $(X_i = 1 \wedge Y_i = 0)$, there exists at least one new position of $\mathbf{M}$ being filled in, i.e., $|G_{i+1}| \geq |G_i| + 1$ and, therefore, after $n = \max(2N', \lambda) \cdot d = \mathsf{poly}(\lambda)$ iterations, $|G_n| \geq N'$. That means the event E only happens if $\sum_{i=1}^{n} X_i \leq \delta \cdot N'$ or there exists at least a round i where $Y_i = 1$. So we have that

$$\Pr[E] \leq \Pr\left[\sum_{i=1}^{n} X_i < \delta \cdot N'\right] + \sum_{i=1}^{n} \Pr\left[Y_i = 1\right]$$

For each round i, we bound $\Pr[Y_i = 1] \leq \frac{\binom{\delta N'}{t}}{\binom{N'}{t}} \leq \delta^t$. Let $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$, we have that

$$\begin{aligned}\Pr\left[\sum_{i=1}^{n} X_i < \delta \cdot N'\right] &\leq \Pr\left[\bar{X} < 1/p' - \left(1/p' - \frac{\delta \cdot N'}{n}\right)\right] \\ &\leq \exp\left(-2n\left(1/p' - \delta \cdot N'/n\right)^2\right) \\ &\leq \exp(-n/p') \leq 2^{-\lambda}\end{aligned}$$

where the second and third inequalities are due to Chernoff-Hoeffding bound and $(p' = \mathsf{poly}(\lambda) \wedge 1/p' < 1/2)$, respectively. Given $t = \mathsf{poly}(\lambda)$, we have that $\Pr[E] \leq n\delta^t + 2^{-\lambda} = \mathsf{negl}(\lambda)$ and this completes the extractability proof.