Received 21 March 2024, accepted 5 April 2024, date of publication 10 April 2024, date of current version 18 April 2024.
Digital Object Identifier 10.1109/ACCESS.2024.3387348

RESEARCH ARTICLE

PoEDDP-A Fast RSA-Based Proof of Possession Accumulator of Dynamic Data on the Cloud

ABDEL ALI HARCHAOUI[1], **ALI YOUNES**[1], **(Member, IEEE), ABDELAAZIZ EL HIBAOUI**[1], **AND AHMED BENDAHMANE**[2], **(Member, IEEE)**
[1]CSSE Laboratory, Faculty of Science, Abdelmalek Essaâdi University, Tetouan 93002, Morocco
[2]LaSAD Laboratory, École Normale Supérieure, Abdelmalek Essaâdi University, Martil 93150, Morocco
Corresponding author: Abdel Ali Harchaoui (a.harchaoui@uae.ac.ma)

ABSTRACT The ease of usage and the convenience of cloud computing come with considerable responsibility. The latter, consists of carefully addressing different security aspects of this technology. The integrity and availability of the outsourced data constitute essential considerations for adopters' final decisions. However, the most critical factor is the efficiency of integrity checks, which must prioritize restricted-resource data owners without affecting the performance of the Cloud Service Provider. This paper proposes a secure scheme, called Proof of Exponentiation of Dynamic Data Possession PoEDDP based on RSA-Accumulators. The proof of concept demonstrates that this scheme is 20 times faster compared to other RSA-based cryptographic accumulator schemes. It could be improved to achieve great results with proper optimizations on the larger integer multiplication side.

INDEX TERMS RSA-Accumulator, cloud storage security, data integrity verification protocol, proof of exponentiation, proof of data possession.

I. INTRODUCTION

At the forefront of modern technology, Cloud Computing provides a vital ecosystem to streamline operations and enhance efficiency across diverse industries. The Everything-as-service (XaaS) model ranging from Infrastructure-as-a-Service to Function-as-a-Service, has become an integral component of the contemporary innovative business landscape [1], [2]. This model features cost-effectiveness, agility, and scalability. Cloud storage accepts a massive amount of data at a rate of terabytes per second, driven by the ease of accessibility, effortless programmability, and seamless integration of various components with cloud platforms, especially from the rapidly growing market of connected resource-restricted devices [3]. However, data integrity is a major concern for adopters when outsourcing data to Cloud Service Providers [4]. Many data owners (generators) cannot trust Cloud Service Providers and their associated third parties to take control of their outsourced data without any guarantees, especially for governments and classified security agencies.

In this paradigm of delegating personal and organizational private data, extensive research has been deployed to secure each component of the cloud ecosystem. Provable Data Possession (PDP) and Proof of Retrievability (PoR) constitute the main integrity verification mechanisms used in this regard. Both mechanisms can be implemented in a deterministic or probabilistic mode. PDP ensures that the data owner can efficiently, periodically, and securely verify the integrity of outsourced data. On the other hand, PoR focuses on proving that the data can be efficiently retrieved from the cloud. While the deterministic mode in both mechanisms ensures total correctness, the probabilistic one only checks the partial correctness of the outsourced data. In scenarios where absolute data integrity is critical, probabilistic methods may fall short of providing the level of assurance required. They introduce an element of uncertainty, making them less robust compared to deterministic ones.

Practically, the Merkle Hash Tree is the most used structure to handle the proof of dynamic data possession [5]. It constitutes a Tree-based cryptographic one-way

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

accumulator used to commit to a set of dynamic data blocks distributed over the Merkle Tree leaves. Nevertheless, as the generated data scales from moderate to very large, the efficiency of Merkle Hash Trees becomes a pressing concern [6]. In response to this challenge, group-based accumulators [7], an alternative cryptographic structure, have gained traction for efficient data handling and their support for fast batch verification [8].

Recently, [9] leveraged group-based accumulators for deterministic integrity check-only in the cloud storage. It proposes a new BlockGen algorithm, using a technique to avoid primality tests and collisions. However, the scheme introduces high computation overhead when performing large integer multiplication and modular exponentiation, using the generated blocks.

To address the partial correctness of the probabilistic PDP, the efficiency problem of, using Merkle Hash Tree, and the drawbacks of the deterministic PDP scheme [9], we propose a deterministic proof of data possession scheme, using RSA-accumulators. Our scheme exploits Proof of Exponentiation and safe under a new stronger security assumption, called the Adaptive Root assumption [10]. Unlike [9], our BlockGen algorithm significantly reduces the computation overhead by reducing the size of the generated blocks used in large integer multiplication. As a consequence, it reduces the communication and storage overhead. Therefore, the scheme is considered to be fast and efficient. It supports data integrity verification characteristics, blockless verification, unrestricted challenge frequency, dynamic data handling, and public auditability. This scheme is secured against different data attacks such as tag forgery attacks, data leakage attacks, replace attacks, replay attacks, and pollution attacks. It can also be deployed on a trusted or untrusted environment in which we regard the accumulator manager's existence or absence.

A. CONTRIBUTIONS

In this paper, we propose a fast and efficient data integrity verification scheme based on a variant of group-based cryptographic accumulators, called RSA-Accumulators, which is safe under the Adaptive Root assumption. It solves the computation, storage, and communication efficiency challenges. The scheme supports different deployment models, such as trusted and untrusted environments in which we consider the accumulator manager malicious and must be restricted in trust-minimized environments where such behaviors may arise. Furthermore, it supports most data integrity verification characteristics, such as blockless verification, unrestricted challenge frequency, dynamic data handling, public auditability, and fairness. Our scheme is secure against different data attacks such as tag forgery attacks, data leakage attacks, replace attacks, replay attacks, and pollution attacks. It maintains data integrity even if the CSP caches some values to accelerate the proof generation without holding the full data. We prove that the accumulator construction is secure, assuming the Strong-RSA.

B. RELATED WORK

The two data structures used in the integrity verification of outsourced data are tree-based and group-based (algebraic) structures. On the one hand, Merkle Hash Tree (MHT) is the most mature tree-based data structure used to authenticate dynamic data and perform remote integrity checks [5]. On the other hand, RSA-based accumulators, also called cryptographic one-way accumulators are considered promising alternatives to Merkle Hash Tree [11]. The commitment in Merkle Hash Tree Fig. 1, is achieved by computing a succinct digest of the set of data blocks distributed over the Merkle Tree leaves, using a Collision Resistant Hash Function. This commitment is used in subsequent checks to verify that a block or a set of blocks indeed still exists. This commitment satisfies a vital security property called binding, which informally means that a malicious adversary cannot produce two or more valid openings once it commits to a set of data.

In this regard, [12] was the first to exploit MHT to maintain the state of dynamic data in the cloud context. Reference [13] introduced proofs-of-ownership (PoWs) to deal with the client-side deduplication based on MHT. Reference [14] PoOR scheme invoked by the client to prove to the CSP its ownership of a particular file and verify its blockless recoverability. It uses erasure codes on an MHT-based structure with homomorphic verifiable tags to achieve deduplication. Reference [15] MuR-DPA is a public auditing scheme based on MHT to support verifiable updates for cloud storage with multiple replicas. This scheme exploits a new Authenticated Data Structure (ADS) called MR-MHT, which organizes replicas for each data block into the same sub-tree. Reference [16] proposed a provable data transfer protocol based on provable data possession and deletion operating on an MHT-based structure. Reference [17] a rank-based Merkle Tree (RBMT) scheme called ODPDP is used to achieve batch updates and outsourced auditing. Reference [18] PRAYS blockless Merkle Tree authenticated structure, with permission-based signature to support collaborative multi-writer. Reference [19] proposed a position-aware structure based on the Merkle Tree to ensure public verifiability of dynamic data without retrieving the whole tree structure to compute the root node. Reference [20] introduced the concept of Dynamic Large Branching Hash Tree (DLBHT) based on MHT combined with a Homomorphic Verifiable Authenticator (HVA) based aggregate signature scheme, to support batch update and public verification.

All the schemes mentioned above use leverage MHT to ensure provable data possession. Thus, they suffer from the efficiency problem reported [6].

Unlike Merkle Hash Tree commitment, group-based cryptographic one-way accumulators provide a succinct witness of inclusion and exclusion against an element independent of the order of elements being accumulated [7], [21]. These proofs are efficient even without a trapdoor [22], [23]. Group-based cryptographic one-way accumulators can be instantiated in different environments, including an untrusted setup where the accumulator manager is considered malicious [24]. They can provide advanced properties like zero-knowledge arguments [25], [26]. In this line of work, Ateniese et al. [27] is the first probabilistic verification-only scheme proposed to ensure that the remote server holds the data of a particular client. The scheme is considered probabilistic and doesn't achieve total correctness. But before that, Caronni and Waldvogel [28] introduced a straightforward deterministic integrity check to establish trust in distributed storage providers. This scheme features public verifiability but is unable to handle dynamic data. It suffers from a significant drawback, requiring the data owner to store replicates of the original data to cross-check the correctness of the received digest. Deswarte et al. [29] presented two methods that use the Diffie-Hellman for key exchange. The first is based on a table containing multiple challenges and precomputed responses for each file, and the second method uses single precomputed value stored on the data owner side for each random challenge. The former method requires the server to be rebooted to compute new responses, which means that this method doesn't achieve unbounded queries. The latter requires modular exponentiation which makes it inefficient for large files. Filho and Barreto [30] is an RSA-based homomorphic hash function scheme proposed to overcome the aforementioned drawbacks. This scheme supports unbounded queries but lacks the support of dynamic data. as well as induces high computation costs. Based on the Diffie-Hellman as well as the RSA hardness, Sebè et al. [31] achieve the auditing without performing exponentiation against the entire data blocks. The proposed scheme shows a tradeoff between the low computation on the CSP side and the storage cost required on the data owner side. This scheme supports unbounded queries but fails to support both dynamic operations on the data and public auditability. Hao et al. [32] proposed a scheme supporting dynamic operation, public verifiability, and privacy against semihonest behavior. The scheme leverages RSA-based homomorphic verifiable tags to support data-level dynamics at the block level. In this construction, when a modification affects the file, the associated blocks and tags are updated, resulting in unnecessary computation and communication expenses. The scheme also suffers from the long time required for the setup and verification phase due to the number of modular exponentiations required. Yi et al. [33] introduced a Provable Data Possession (PDP) scheme utilizing Fully Homomorphic Encryption (FHE) with multi-copy support to ensure public verifiability of data duplicates while preserving confidentiality from third-party authenticators. The scheme enables dynamic data support and authorizes user access to shared data. This scheme is susceptible to potential decryption errors when multiple homomorphic operations are performed, leading to increased noise in the ciphertext. Additionally, there is a drawback in storage costs arising from data expansion in the FHE process. Recently, [9] relied on a modified RSA-group of known order, which is safe under the strong RSA assumption to build a

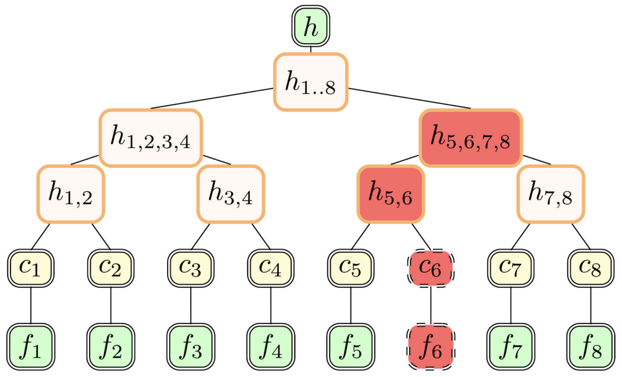

h
$h_{1..8}$
$h_{1,2,3,4}$
$h_{5,6,7,8}$
$h_{1,2}$
$h_{3,4}$
$h_{5,6}$
$h_{7,8}$
c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8
f_1 f_2 f_3 f_4 f_5 f_6 f_7 f_8

FIGURE 1. A merkle tree commitment.

variant of group-based cryptographic one-way accumulators called RSA-Accumulator. Based on these assumptions, they construct a deterministic data integrity verification scheme in the cloud context that supports dynamic data. They introduce a non-coprime representation of the precomputed data blocks, using a BlockGen algorithm, first to avoid primality tests and collisions, second to overcome the computation overhead of the large modular exponentiations of inputs, and finally to prevent the CSP from issuing a proof without holding the data. The scheme supports public auditability, dynamic operation, privacy preserving, and more. However, the scheme suffers from high computation overhead during both the block generation and proof generation phases due to the length of the generated blocks used during the large integer multiplication and modular exponentiation. Ren et al. [34] proposed a deterministic integrity verification scheme that utilizes another variant of group-based cryptographic one-way accumulators called bilinear map accumulators [26]. This scheme is safe under a Strong Diffie Hellman assumption noted $\mathsf{t-SDH}$. Thus, the scheme is restricted with an upper bound of t elements to be accumulated.

C. ORGANIZATION OF THE PAPER

Our paper is organized as follows: Section II equipped the reader with the required background, definitions, assumptions, and the adopted notations used throughout the paper. Section III describes our proposed scheme from a high-level overview. Sections IV and V detail the building blocks of the scheme. Section VI highlights the characteristics of our scheme, and VII defines the security requirements, provides formal proofs, and performs a performance analysis. Section IX discusses the results of our scheme and the Section X concludes the paper.

II. PRELIMINARIES

Throughout this section, we will highlight some background and definitions, the adopted notations used in the paper, and the assumptions that we built upon.

A. NOTATIONS

- $\mathsf{negl}(\lambda)$ is a negligible function of the security parameter λ.

- $\mathsf{Primes}(\lambda)$ is the set of odd primes less than 2^{λ}.
- $\chi \xleftarrow{\$} \mathbb{G}$ denotes a uniform sampling of the element χ from the set $\mathbb{G}$.
- $\chi \xleftarrow{\$} \mathcal{A}(.)$ is a randomized algorithm $\mathcal{A}$ that produces a random variable χ.
- $\mathsf{GGen}(\lambda)$ is a randomized algorithm that generates either a group of known order or a group of unknown order.
- $\mathsf{keyGen}(\lambda)$ is a key generation algorithm that produces the verification key vk and public key pk.

B. BACKGROUND

1) CRYPTOGRAPHIC-ACCUMULATORS

Cryptographic accumulators, as introduced in [11], were essentially used as a one-way hash function f with the property of being quasi-commutative primitives. A quasi-commutative function f is a function such that for all initial value $g \in \mathbb{G}$ and for all elements $x_1, x_2 \in X$:

$$\begin{aligned} &f : \mathbb{G} \times X \longrightarrow \mathbb{G} \\ &\forall (g, x_1, x_2) \longrightarrow f(f(g, x_1), x_2) = f(f(g, x_1), x_2) \end{aligned} \tag{1}$$

These cryptographic primitives are designed to produce a succinct binding commitment $A_X = f(f(f(\ldots f(f(f(g, x_1), x_2), x_3), \ldots, x_{n-2}), x_{n-1}), x_n)$ where $X = \{x_1, \ldots, x_n\}$ is the set of elements. They generate short membership and non-membership proof to test whether an element x_i is in the set X or not. These accumulators can be constructed, using different structures. One employs the hash-based (or tree-based) approach [5], [35], utilizing the Merkle tree structure. The other uses the group-based (or algebraic-based) approach, exploiting the algebraic structure of the underlying groups. Furthermore, the group-based accumulators can be further categorized into RSA-Accumulators [7], [21], [24], [36], built on top of the RSA-groups (Definition 1), and Bilinear-Accumulators [37], [38], [39] built, using bilinear pairings. In this work, our focus will be on the RSA-Accumulators.

The simplest scenario of using these primitives is when a prover possesses a set of elements $x_i \in X$ and wants to prove to a verifier that he holds an element x_k. So the prover issues a proof π_k to express that he holds the element in question, then the verifier executes a check against this proof π_k to verify the claim. Technically, given a group $\mathbb{G}$ and a random element $g \in \mathbb{G}$ such that $\mathbb{G}$ is an RSA-group defined by multiplying to big prime numbers p, q such that $N = p \cdot q$ (as mentioned in next section 1). The prover computes a binding commitment A_X to the set X, using modular exponentiation: $A_X = g^{x_1 . x_2 \ldots x_{k-1} . x_k . x_{k+1} \ldots x_n} \in \mathbb{G}$. This commitment can be publicly disclosed for (non)membership verification. A verifier who wants to audit for example if $x_k \in X$ challenges the prover. The prover then generates a proof π_k that x_k is indeed in X by computing the same modular exponentiation, using all elements x_i except for x_k. The verifier then checks the proof to determine if ${\pi_k}^{x_k}$ is equal to A_X.

Definition 1 (RSA-Groups): We call RSA-group the group $\mathbb{Z}_N^*$, i.e., the multiplicative group of invertible integers modulo N, where N is the product of two big primes. We define the RSA quotient group for N as the group $\mathbb{Z}_N^*/\{\pm 1\}$. It is believed that all elements of $\mathbb{Z}_N^*/\{\pm 1\}$ are of unknown order except for the identity element.

Definition 2 (Collision-Resistant hash Function): Let $H : \{0, 1\}^* \rightarrow \{0, 1\}^{\lambda}$ be a hashing function, we say that H is collision-resistant if the advantage of a PPT adversary $\mathcal{A}$ to find $x_1 \neq x_2 \wedge H(x_1) = H(x_2)$ is negligible.

Definition 3 (Division-Intractable Hashing Function): Let $H : \{0, 1\}^{\lambda} \rightarrow \{0, 1\}^{\lambda}$ be a collision-resistant hash function, H is division-intractable if the advantage of PPT adversary $\mathcal{A}$ to find and element x_j and a set of $\{x_i\} \in \{0, 1\}^{\lambda}$ and $H(x_j)$ divides $\prod_{i=1}^{n} H(x_i)$ is negligible.

2) PROOF OF EXPONENTIATION

Proof of Exponentiation PoE is a protocol that allows a prover to convince a verifier that he knows the correct value α such that $\alpha = g^{\tau}$, where g and τ are known to the verifier. It offers efficient verification for resource-intensive computation (i.e., iterated exponentiation) and allows the outsourcing of this computation when the verifier is resource-restricted.

The iterated exponentiation is the commonly used strategy by most RSA-Accumulators to compute the commitment A_X and the proof π_k, on the one hand, and the verification for a given element x_k, on the other hand. However, this strategy is time and resource-intensive. In our case, the PoE protocol leverages a shortcut for resource-constraint verifiers to efficiently check the correct computation of the exponentiation without performing the whole computation.

Technically, let the shared inputs g, α in $\mathbb{G}$ and x^* in $\mathbb{Z}$ such that $x^* = \prod_{i=1}^{n} x_i$, if the CSP wants to convince the data owner that he is still holding all the elements x_i's, and the computation of $\alpha = g^{x^*}$ was done correctly. First, the data owner sends a random prime number l from $\mathsf{Primes}(\lambda)$ to the CSP. Then, the CSP finds the pair (q, r) such that q in $\mathbb{Z}_{\mathbb{N}}$ and r in $[l]$ where $x^* = q \cdot l + r$ in $\mathbb{G}$, and sends back $\pi = g^q$. Finally, the data owner checks if $\pi^l g^r = \alpha$ holds in $\mathbb{G}$. The latter protocol is the Proof of Exponentiation PoE protocol introduced by Wesolowski [10] and generalized by Boneh et al. [8]. One can notice that this protocol holds significant utility for the CSP, and it proves highly advantageous when the data owner is resource-restricted. The job of the former is reduced from recomputing the whole iterated exponentiations (g^{x^*}) into performing one Euclidean division of x^* by l and one exponentiation g^q. The latter's job is reduced from the old computation cost into one Euclidean division of x^* by l and one comparison $g^{q \cdot l + r} = g^{x^*}$ in $\mathbb{G}$. This is considered faster compared to recomputing the iterated exponentiation g^{x^*}.

3) ADAPTIVE ROOT ASSUMPTION AND STRONG-RSA ASSUMPTION

The soundness of our PoEDDP relies on a new computationally hard assumption denoted as the Adaptive Root Assumption (Definition 5), introduced in [10] and studied in [40] and [41]. This assumption allowed our scheme to be instantiated in more general groups, where it is believed to hold (i.e., the cyclic group $\mathbb{Z}_N/\pm 1$ and class groups). The basic idea behind this assumption is that it is hard to extract the roots of a given element adaptively. Adaptive in the context of security models allows the adversary to behave dynamically and adapt his strategies based on information acquired during the extraction process of the root.

Let say that we want to prove that $\alpha = g^{\tau}$, using the Proof of Exponentiation as mentioned above and in II-B2 such that τ is the product of the elements $x_i \in X$. The verifier sends a random prime $l \in [0, 2^{\lambda}]$ and verifies the correctness of the proof π, using the following check $\pi^l g^r = \alpha$ such that $r = \tau(\mod l)$, where $\pi = g^q$ is computed by the prover such that $\tau = q \cdot l + r$. If an adversary fixes some arbitrary values $\hat{g} \in \mathbb{G}$ and $\hat{\alpha} \neq 1$, once he gets the sampled challenge l from the verifier, he will try to adaptively forge a valid proof $\hat{\pi}$ such that $\hat{\alpha} \cdot \hat{g}^{\hat{\tau}} = \hat{\pi}^l \cdot \hat{g}^r$, and $r = q \cdot l + r$. He can do so by finding an element $\hat{u}$ such that $\hat{\alpha} = \hat{u}^l$. If this extraction is done correctly, then $\hat{\pi} = \hat{\alpha}^{\frac{1}{l}} \cdot \hat{g}^q$ is considered as a valid proof. Thus, the adversary is able to find the l-th root of $\hat{\alpha}$ for an arbitrary non trivial element l sent by the verifier. However, this computation is considered to be hard in the groups of interest. Opposed to the Adaptive Root Assumption which is still developing and a non-standard assumption, the Strong-RSA Assumption (Definition 6) is classical and well established assumption, introduced in [36]. It states that it is difficult to compute the l-th root of a random RSA modulus N, given only the value of l, where l is a random odd prime.

To wrap up, the difference between the Adaptive Root Assumption and the Strong-RSA Assumption is that the former is less restrictive regarding group selection and asserts that it is hard to extract a random root of a chosen group element. However, the latter is more restrictive regarding group selection and asserts that it is hard to extract a chosen root of a random group element. For more formal definitions and technical proofs, one can check 5 and VII, respectively.

4) ACCUMULATOR SECURITY (UNDENIABILITY)

We follow the undeniability property as introduced in [24] to define the security of the accumulator. This property ensures the accumulator's security and integrity. It prevents the accumulator manager from simultaneously confirming and denying the presence of a specific element if he intends to act maliciously. Technically, if a particular element x_k doesn't exist among a set of elements X accumulated into an accumulated value A_X, an adversary $\mathcal{A}$ cannot later provide a proof of inclusion π_k of the element x_k in the accumulator A_X and provide, at the same time, another proof of exclusion $\hat{\pi_k}$ to prove the opposite.

In this work, we adapt the formal definition of the undeniability as appeared in [24] and generalized in [42] to follow our notations. *ProofVer* is an algorithm taking the accumulated value α_t, the element x_k and the proof π_k or $\hat{\pi_k}$ as inputs and outputs {0,1} to accept or refuse the provided proof. We say that the accumulator is undeniable if for all PPT adversary $\mathcal{A}$ the following advantage is negligible considering the Strong-RSA assumption:

Definition 4 (Accumulator Security (Undeniability)):

$$Pr\left[\begin{array}{c} \mathbb{G} \xleftarrow{\$} GGen(\lambda) \\ g \xleftarrow{\$} \mathbb{G} \\ (\alpha_t, x_k, \pi_k, \hat{\pi_k}) \leftarrow \mathcal{A}(\mathbb{G}, g) \\ ProofVer(\alpha_t, x_k, \pi_k) = 1 \\ \wedge ProofVer(\alpha_t, x_k, \hat{\pi_k}) = 1 \end{array}\right] = \mathsf{negl}(\lambda) \qquad (2)$$

5) TRUSTED AND UNTRUSTED SETUP

The trusted setup requires private randomness p and q to generate the public parameters $N = p \cdot q$ and g, necessary to our scheme. These private randomness must be kept secret or discarded to maintain the soundness of our scheme. If they are disclosed to an adversary or the CSP wants to act maliciously, it breaks the binding property of our scheme to forge arbitrary proofs. The forgery of such proofs could be easily constructed for any element x_k as $\pi_k = g^{\left(x^* \cdot \frac{1}{x_k}\right)(\mod \phi(N))}(\mod N)$, where $\phi(N) = (p-1)(q-1)$. Thus, the existence of a trusted party (other than the CSP), which efficiently generate this RSA modulus, using GGen and does not disclose the private randomness, is crucial. Alternatively, GGen can use public randomness to select a sufficiently large N during the trusted setup, making factoring N challenging. However, the larger N is, the impractical it becomes. Often, trusted parties are undesirable in trust-minimized environments where malicious behaviors may arise. Thus, the reliance on a single trusted party can be reduced as in [43], [44], [45], [46], and [47] by engaging multiple parties in the setup process or by employing decentralized methods. This choice between trusted and untrusted setups involves a tradeoff between efficiency, complexity, computation overhead, and security. Trusted setups offer simplicity and efficiency but introduce a central point of trust. On the other hand, untrusted setups distribute trust among multiple parties or eliminate them completely to enhance security, but often at the cost of increased computational complexity. Hence, striking the right balance depends on the specific application's requirements and the desired level of security assurance.

C. ASSUMPTIONS

Definition 5 (Adaptive Root Assumption): Let GGen be the generator of the group $\mathbb{G}$ of a security parameter λ. We say that the Adaptive Root Assumption for GGen holds if the advantage of an adversary $(\mathcal{A}_0, \mathcal{A}_1)$ to: first, $\mathcal{A}_0$ outputs an element $w \in \mathbb{G}$ and some state **st**. Then, a randomly chosen prime l in $\mathsf{Primes}(\lambda)$ and $\mathcal{A}_1(w, l, st)$ outputs $w^{\frac{1}{l}} \in \mathbb{G}$ is negligible. For all efficient ($\mathcal{A}_0, \mathcal{A}_1$):

$$Pr\left[\begin{array}{r} \mathbb{G} \xleftarrow{\$} GGen(\lambda) \\ (w, st) \leftarrow \mathcal{A}_0(\mathbb{G}) \\ u^l = w \neq 1 : l \xleftarrow{\$} \mathsf{Primes}(\lambda) \\ u \leftarrow \mathcal{A}_1(w, l, st) \end{array}\right] \leq \mathsf{negl}(\lambda) \quad (3)$$

Definition 6 (Strong-RSA Assumption): GGen satisfies the strong RSA assumption if for all efficient $\mathcal{A}$:

$$Pr\left[\begin{array}{lr} & \mathbb{G} \xleftarrow{\$} GGen(\lambda), \\ & g \xleftarrow{\$} \mathbb{G} \\ \alpha^e = g & \\ \text{s.t } e \text{ is odd prime} & : (\alpha, e) \in \mathbb{G} \times \mathbb{Z} \xleftarrow{\$} \mathcal{A}(\mathbb{G}, g) \end{array}\right]$$
$$\leq \mathsf{negl}(\lambda) \quad (4)$$

III. OVERVIEW

In this section, we briefly describe our proposed deterministic data integrity verification scheme and its three major phases as shown in Fig. 2.

Our scheme relies on recent results published by Wesolowski [10], called Proof of Exponentiation PoE. We exploit PoE over an RSA-Accumulator to build a secure scheme, called Proof of Exponentiation of Dynamic Data Possession PoEDDP. This scheme supports dynamic data, using uniquely computed tags inserted into a persistent dynamically indexed map, called Hash-to-Prime Mapping (HPM). The scheme also supports fast proof verification, low computation, and communication overhead as shown in Figures 16, 17, and 18. That makes it suitable for low and restricted resources devices. It supports public auditability, blockless verification, and unbonded queries. It fulfills completeness and soundness properties when the chosen group satisfies the Adaptive Root Assumption and provides fairness against dishonest verifiers.

1) The setup phase: creates the scheme environment with a security parameter λ and the desired deployment model II-B5. It executes the $\mathsf{GGen}(\lambda)$ algorithm to output the necessary public parameters as follows: a) a group $\mathbb{G}$ of order N, using private randomness p and q if one opts for a trusted setup or public randomness if one chooses the untrusted setup. $\mathsf{GGen}(\lambda)$ can also produce a group $\mathbb{G}$ of unknown order for both cases too, b) a generator g of the group $\mathbb{G}$. In addition to the public parameters, the keyGen algorithm produces the public key pk and verification key vk.
2) The preprocessing phase: the data owner splits the data $\mathcal{D}$ into n-blocks f_i of size l_1, using the split algorithm and pads it if necessary. This set of data is defined as $\mathcal{F} = \{f_i\}_{i=1}^n$. Then, encrypts each block f_i with a secure cipher Enc to produce the ciphertext c_i of size l_2 and form a set of encrypted blocks $\mathcal{C} = \{c_i\}_{i=1}^n$. To finalize the preprocessing phase, the ciphertext c_i is fed to the $\mathsf{H}_{\mathsf{prime}}$ algorithm to produce odd primes τ_i of size l_3. These tags are saved into HPM for a later usage. Later on, he computes the accumulated value α_t, using this set of odd primes τ_i such that $\alpha_t = g^{\prod_{i=1}^n \tau_i}$. Then, he stores α_t to be used during the verification phase, outsources the concatenation ($\mathcal{C} \;||\; N \;||\; g$) of the ciphertexts, the group description and the generator to the CSP, and deletes his local version of the data $\mathcal{F}$.
3) During the auditing phase (Challenge/Verification phase): the data owner has to ensure that the CSP holds the outsourced data and maintains its authenticity. He starts the verification protocol by sending a challenge to the CSP. This challenge consists of a randomly chosen prime number $l \xleftarrow{\$} \mathsf{Primes}(\lambda)$, and a randomly chosen index j from HPM. Then, the role of the CSP is to convince the data owner. First, he searches for the ciphertext of the index j (c'_j), hashes it to prime to get τ'_j, then tries to find a pair (q, r) such that $q \in \mathbb{Z}_{\mathbb{N}}$ and $r \in [l]$ where $\prod_{i=1}^{n} \tau'_i \times \tau_j'^{-1} = q \cdot l + r$, and τ'_j is the hash-to-prime of c'_j done by the CSP. Finally, he sends back the concatenation ($\pi \;||\; c'_j$) of the proof of data possession $\pi = g^q$ and the witness c'_j to the data owner. To end the verification phase, the data owner compares the hash-to-prime of the received c'_j with his own τ_j and tries to find a value $r = \prod_{i=1}^{n} \tau_i \times \tau_j^{-1} (\mod l)$ and accepts if $\pi^l g^r = \alpha_t^{\tau_j^{-1}}$ holds.

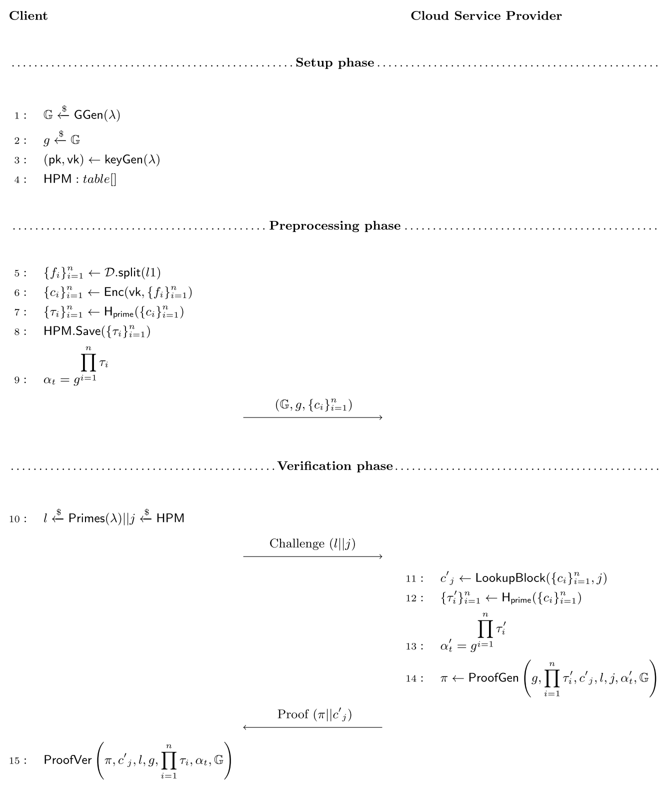
IV. CONSTRUCTIONS

This section describes the building blocks of PoEDDP scheme and details the procedures used during its phases.

A. PHASES

1) SETUP PHASE

Given a security parameter λ which specifies the required security degree of the RSA key size (i.e., 3072 RSA key size for 128 bits of security). In Fig. (3), the data owner, denoted as V for verifier, generates a group $\mathbb{G}$, using the GGen algorithm. There exist two possible choices: 1) If we opt for a completely untrusted environment, the GGen algorithm is run, using decentralized methods: an untrusted setup model via Multi-Party Computation (MPC) or threshold method to produce necessary secrets to build group $\mathbb{G}$. Otherwise, $\mathbb{G}$ of an unknown order can be chosen (i.e., class groups). 2) If the data owner is an accumulator manager playing the role of a trusted entity, which could affect the fairness property toward the CSP, the GGen algorithm produces a group $\mathbb{G}$ of known order $N = p \cdot q$ such that p and q are two relatively big primes (i.e., $\mathbb{Z}_{\mathbb{N}}^*$). Then, we sample a random element g from $\mathbb{G}$ used to instantiate our accumulator. Finally, the data owner instantiates a mapping of the accumulated values called HPM, which consists of a dynamic indexed mapping used to track the elements involved to generate the



```
Client                                              Cloud Service Provider

.......................... Setup phase ..........................

1 :  $\mathbb{G} \xleftarrow{\$} \mathsf{GGen}(\lambda)$
2 :  $g \xleftarrow{\$} \mathbb{G}$
3 :  $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{keyGen}(\lambda)$
4 :  $\mathsf{HPM} : table[]$

.......................... Preprocessing phase ..........................

5 :  $\{f_i\}_{i=1}^n \leftarrow \mathcal{D}.\mathsf{split}(l1)$
6 :  $\{c_i\}_{i=1}^n \leftarrow \mathsf{Enc}(\mathsf{vk}, \{f_i\}_{i=1}^n)$
7 :  $\{\tau_i\}_{i=1}^n \leftarrow \mathsf{H_{prime}}(\{c_i\}_{i=1}^n)$
8 :  $\mathsf{HPM.Save}(\{\tau_i\}_{i=1}^n)$
9 :  $\alpha_t = g^{\prod_{i=1}^n \tau_i}$

              $\xrightarrow{(\mathbb{G}, g, \{c_i\}_{i=1}^n)}$

.......................... Verification phase ..........................

10 : $l \xleftarrow{\$} \mathsf{Primes}(\lambda) || j \xleftarrow{\$} \mathsf{HPM}$

              $\xrightarrow{\text{Challenge } (l||j)}$

                                 11 : $c'_j \leftarrow \mathsf{LookupBlock}(\{c_i\}_{i=1}^n, j)$
                                 12 : $\{\tau_i'\}_{i=1}^n \leftarrow \mathsf{H_{prime}}(\{c_i\}_{i=1}^n)$
                                 13 : $\alpha_t' = g^{\prod_{i=1}^n \tau_i'}$
                                 14 : $\pi \leftarrow \mathsf{ProofGen}\left(g, \prod_{i=1}^n \tau_i', c'_j, l, j, \alpha_t', \mathbb{G}\right)$

              $\xleftarrow{\text{Proof } (\pi||c'_j)}$

15 : $\mathsf{ProofVer}\left(\pi, c'_j, l, g, \prod_{i=1}^n \tau_i, \alpha_t, \mathbb{G}\right)$
```

FIGURE 2. PoEDDP scheme.

Setup(λ):

1) The data owner V generates two relatively big prime numbers p,q such that $p \neq q$ (i.e., of size 1536 bits to achieve λ=128 bits of security).
2) V multiplies p and q to produce a group $\mathbb{G}$ order N.
3) V chooses a random element g from the interval [0, N-1].
4) V instantiates a dynamic mapping HPM to hold the accumulated values.

FIGURE 3. Setup phase.

accumulated value α_t in a given time t. To perform later data encryption, with the secure Enc cipher, we generate the (pk, vk), using the keyGen algorithm.

2) PREPROCESSING PHASE

In this phase shown in Fig. 4, the data owner V preprocesses the data he wanted to outsource to the CSP denoted as P for prover.

1) He splits the data file $\mathcal{D}$ into n-blocks f_i of size l_1, using split procedures. If the block f_i is not a multiple of l_1, V pads it accordingly.
2) Then, he encrypts each block f_i, using a secure block cipher Enc (i.e, AES) with the key vk to produce the encrypted block c_i of size l_2.
3) He hashes to prime each block c_i using a hash-to-prime division-intractable hashing function $\mathsf{H_{prime}}$ to generate hashes to primes τ_i of size l_3 linked to each block c_i,
4) and stores τ_i into the dynamic mapping HPM.
5) Then, V computes the digest $\alpha_t = g^{\prod_{i=1}^n \tau_i}$, which is the sampled element g raised to the power of the product of the hashes to prime of the encrypted blocks. Finally,
6) V outsources the encrypted data $\mathcal{C}$ to P, concatenated with the public parameters N and g as ($\mathcal{C}$ || N || g), using a secure channel (i.e., TLS 1.3).

3) AUDITING PHASE

During this phase Fig. 5, V initiates the protocol to verify that P holds the full version of the outsourced data, by the mean of the accumulated value α_t that he owned:

Preprocessing:
V prepares the data $\mathcal{D}$ to be outsourced:

1) he splits it into blocks f_i of size l_1; $\mathcal{F} \leftarrow \mathcal{D}.\mathsf{split}(l_1)$,
2) encrypts each block f_i into c_i; $c_i \leftarrow \mathsf{Enc}(\mathsf{vk}, f_i)$ such that $f_i \in \mathcal{F}$ and $c_i \in \mathcal{C}$,
3) hashes to prime each block c_i into τ_i; $\tau_i \leftarrow \mathsf{H_{prime}}(c_i)$ such that $\tau_i \in \mathcal{T}$
4) saves the tags into the HPM mapping' $\mathsf{HPM.Save}(\tau_i)$ such that $\tau_i \in \mathcal{T}$,
5) computes the accumulated value $\alpha_t = g^{\prod_{i=1}^{n} \tau_i}$ such that $\tau_i \in \mathcal{T}$ and saves α_t on his side.
6) Then, outsources the concatenation $(\mathcal{C} \mid\mid N \mid\mid g)$ of the encrypted data $\mathcal{C}$, the public parameters N and g to P and deletes his local version of the data.

FIGURE 4. Preprocessing phase.

1) he randomly generates a prime number $l \overset{\$}{\leftarrow} \mathsf{Primes}(\lambda)$, and a random index j from HPM, then, sends them to P as a challenge to P.
2) P receives $(l \mid\mid j)$, he searches for the block c'_j to compute his version of the hashing to prime τ'_j. He then tries to find the pair (q, r) such that q in $\mathbb{Z}_\mathbb{N}$ and r in $[l]$ where $\prod_{i=1}^{n} \tau'_i \times \tau'^{-1}_j = q \cdot l + r$, and τ'_i is the hash-to-prime of the encrypted block c_i performed by P. Then, P proves the possession of V's data by computing $\pi = g^q$ and sending the concatenation of the proof π and the witness c'_j to V. Finally, V gets $(\pi \mid\mid c'_j)$, compares $\tau'_j \leftarrow \mathsf{H_{prime}}(c'_j)$ with his local τ_j. Then, he tries to find the value r such that $r = \prod_{i=1}^{n} \tau_i \times \tau'^{-1}_j (\mod l)$, and accepts if $\pi^l g^r = \alpha_t^{\tau'^{-1}_j}$ following the proof of correctness of our protocol described in (5):

$$\pi^l g^r = \alpha_t^{\tau'^{-1}_j} \tag{5}$$

$$\Leftrightarrow \quad < \ def(\pi)^l = (g^q)^l, def \alpha_t = g^{\prod_{i=1}^{n} \tau_i} \ >$$

$$(g^q)^l g^r = g^{\left(\prod_{i=1}^{n} \tau_i\right)^{\tau'^{-1}_j}}$$

$$\Leftrightarrow \quad < \text{ exponents identities } >$$

$$g^{ql+r} = g^{\prod_{i=1}^{n} \tau_i \times \tau'^{-1}_j}$$

$$\Leftrightarrow \quad < \text{ exponents property } >$$

$$ql + r = \prod_{i=1}^{n} \tau_i \times \tau'^{-1}_j$$

Q.E.D

V. DATA DYNAMICS SUPPORT

We have just constructed our initial RSA-based Provable Data Possession accumulator, denoted α_t, over the set of the hashes to primes τ_i's stored in HPM table. Handling static data carries little significance, especially in light of the radical transformation introduced by IoT and edge/cloud computing. These devices capture and generate a vast amount of data, which may contain sensitive or critical information. This data undergoes processing, storage, updates, long-term archiving, or deletion. This dynamic nature, along with integrity, confidentiality, and privacy are of paramount consideration for different organizations when they decide to outsource their sensitive data to cloud computing and transiting to full-cloud entities. Thus, our PoEDDP leverages the necessary mechanism to ensure the security of the dynamic nature of the data without sacrificing the efficiency. In this section, we describe our approach for dynamic data handling to support various operations. We begin by addressing the simpler cases of single-block addition, deletion, and update. Subsequently, we expand the scheme to accommodate m-blocks operations.

A. ADD ONE BLOCK

Let's say that the data owner wants to add a new block f_k to the cloud storage, which holds the old data $\mathcal{F}$ on its encrypted form $\mathcal{C}$. Fig. 6 shows the state of the HPM mapping on the data owner side and the state of the encrypted blocks $\mathcal{C}$ on the CSP side, before and after adding the block of data f_k.

1) The data owner V creates one block denoted f_k of size l_1 such that k is the position where the new block will be added. He pads the block f_k if the length of $f_k < l_1$, performs the encryption, using the cipher Enc, and the authentication, using $\mathsf{H_{prime}}$ to produce the pair (c_k, τ_k) of the ciphertext c_k and the tag τ_k. Then, he stores the hash-to-prime into the HPM, $\mathsf{HPM.Save}(\tau_k)$, and commits to the new set $\mathsf{HPM} \cup \{\tau_k\}$, by computing the new digest $\alpha_{t+1} = \alpha_t^{\tau_k}$. This quantity represents the old digest, raised to the power of the hash-to-prime of the new encrypted block c_k. In the end, V outsources the

Challenge:

1) V chooses $l \xleftarrow{\$}$ Primes(λ), and an index j
2) sends the concatenation $(l \mid\mid j)$ to P.

ProofGen:

Once P receives $(l \mid\mid j)$:

1) he lookups for the block of index j, using LookupBlock; $c'_j \leftarrow$ LookupBlock$(\mathcal{C}, j)$,
2) computes $\tau'_i \leftarrow \mathsf{H}_{\mathsf{prime}}(c_i)$ for each block c_i such that $c_i \in \mathcal{C}$,
3) tries to find (q, r) such that $q \in N$ and $r \in [l]$ where $\prod_{i=1}^{n} \tau'_i \cdot \tau'^{-1}_j = q \cdot l + r$ such that $\tau'_j \leftarrow \mathsf{H}_{\mathsf{prime}}(c'_j)$
4) produces the proof $\pi \leftarrow g^q$, and
5) sends the concatenation $\left(\pi \mid\mid c'_j\right)$ of the proof and the witness to V.

ProofVerify:

1) V receives $\left(\pi \mid\mid c'_j\right)$, he then verifies and accepts (output 1 or 0 otherwise) if:

$$\begin{cases} \tau_j = \tau'_j \leftarrow \mathsf{H}_{\mathsf{prime}}(c'_j) \\ r \leftarrow \prod_{i=1}^{n} \tau_i \cdot \tau'^{-1}_j (\mod l) \\ \pi^l g^r = (\alpha_t)^{\tau_j^{-1}} \end{cases} \tag{6}$$

FIGURE 5. Auditing phase.

new encrypted block c_k to the P, concatenated with a randomly chosen prime $l \xleftarrow{\$}$ Primes(λ), and an index j, to challenge P as $(c_k \mid\mid l \mid\mid j)$.

2) Upon receiving $(c_k \mid\mid l \mid\mid j)$, P computes his own hash-to-prime τ'_k of c_k and inserts c_k into k-th position of $\mathcal{C}$. Then, he proceeds to the proving process to prove the correct insertion of the newly received block. He searches for the encrypted block c'_j of the received index j to compute the corresponding hash-to-prime τ'_j. He then tries to find the pair (q, r) such that $q \in \mathbb{Z}_{\mathbb{N}}$ and $r \in [l]$ where $\prod_{i=1}^{n} \tau'_i \cdot \tau'^{-1}_j = q \cdot l + r$. Finally, P computes $\pi = g^q$, and sends the concatenation $\left(\pi \mid\mid c'_j\right)$ of the proof and the witness to V.
3) V receives $\left(\pi \mid\mid c'_j\right)$ and accepts the claim that the block f_k was inserted correctly and P still has the complete data $\mathcal{C}$ if equation (7) holds, following the same proof of correctness previously done in (5).

$$\begin{cases} \tau_j = \tau'_j \leftarrow \mathsf{H}_{\mathsf{prime}}(c'_j) \\ r \leftarrow \prod_{i=1}^{n} \tau_i \cdot \tau'^{-1}_j (\mod l) \\ \pi^l g^r = (\alpha_{t+1})^{\tau_j^{-1}} \end{cases} \tag{7}$$

B. ADD MULTIPLE BLOCKS

We expanded the previous process, of adding 1-block and proving its correct outsourcing to the remote CSP, to support batch addition of m-blocks of data, as described in Fig. 8.

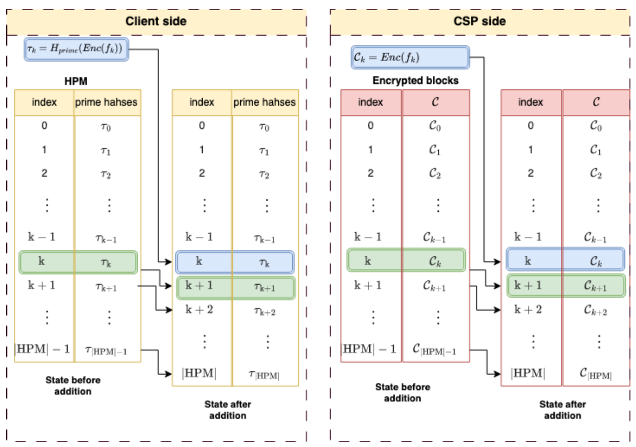


FIGURE 6. The state of the HPM table on the client side and encrypted blocks on the CSP side before and after adding one element.

Let $\mathcal{D}_m$ be the set of the new unique m-blocks, that the data owner wanted to outsource, $\mathcal{F}_m$ be the set of the m-padded blocks of size l_1 of $\mathcal{D}_m$, $\mathcal{C}_m$ be the set of the m-ciphertexts of each element of $\mathcal{F}_m$, and $\mathcal{T}_m$ be the set of the m-hashes-to-prime created from each element of $\mathcal{C}_m$. The data owner V:

1) prepares the m-blocks of $\mathcal{D}_m$, using the split procedure to create blocks f_j of size l_1, and performs padding if $f_j < l_1$, where $j = \{1, \ldots, m\}$,
2) encrypts each block f_j, using the cipher Enc to produce c_j,
3) authenticates the each ciphertext c_j, using the $\mathsf{H}_{\mathsf{prime}}$ procedure to produce the tags τ_j,

Preprocessing:
V prepares the block f_k to add:

1) V encrypts f_k; $c_k \leftarrow \mathsf{Enc}(\mathsf{vk}, f_k)$
2) hashes to prime c_k; $\tau_k \leftarrow \mathsf{H_{prime}}(c_k)$,
3) saves the new tag into the HPM; HPM.$save(\tau_k)$,
4) computes the new accumulator as $\alpha_{t+1} \leftarrow {\alpha_t}^{\tau_k}$ and saves it locally,
5) chooses $l \overset{\$}{\leftarrow} \mathsf{Primes}(\lambda)$ and a random index j from HPM as a challenge.
6) Then, outsources the encrypted block and the challenge; $(c_k \mid\mid l \mid\mid j)$.

ProofGen:
Once P receives $(c_k \mid\mid l \mid\mid j)$:

1) he finds $c'_j \leftarrow \mathsf{LookupBlock}(\mathcal{C}, j)$,
2) computes $\tau'_j \leftarrow \mathsf{H_{prime}}(c'_j)$ of the witness c'_j,
3) computes $\tau'_k \leftarrow \mathsf{H_{prime}}(c_k)$ of the new received block,
4) inserts c_k into the correct index k.
5) tries to find $(q, r) \in \mathbb{Z}_\mathbb{N} \times [l] : \prod_{i=1}^{n} \tau'_i \cdot {\tau'_j}^{-1} = q \cdot l + r$,
6) produces the proof $\pi \leftarrow g^q$,
7) sends the concatenation $\left(\pi \mid\mid c'_j\right)$ of the proof and the witness to V.

ProofVerify:

1) V receives $(\pi \mid\mid c'_j)$, he then verifies and accepts if

$$\begin{cases} \tau_j = \tau'_j \leftarrow \mathsf{H_{prime}}(c'_j) \\ r \leftarrow \left(\prod_{i=1}^{n} \tau_i \cdot {\tau'_j}^{-1} (\mod l)\right) \\ \pi^l g^r = (\alpha_{t+1})^{\tau_j^{-1}} \end{cases} \tag{8}$$

FIGURE 7. One block addition.

4) saves each hash-to-prime τ_j into the dynamic mapping HPM,
5) computes the new accumulated value α_{t+1}, which is the old accumulator α_t raised to the power of the product of the new m-hashe-to-prime of the encrypted blocks f_j, as follows $\alpha_{t+1} = {\alpha_t}^{\prod_{j=1}^{m} \tau_j}$,
6) outsources the new encrypted data $\mathcal{C}_m$ concatenated with a randomly chosen prime l and an index k from the HPM as $(\mathcal{C}_m \mid\mid l \mid\mid k)$ to the CSP. l and k will be used as a challenge to produce the proof of correct insertion.

Once the CSP receives $(\mathcal{C}_m \mid\mid l \mid\mid k)$. He appends the m-blocks to the old encrypted data $\mathcal{C}$ and tries to produce a proof to convince V that the insertion was done correctly. Consequently, he follows the same process described previously by retrieving the encrypted block c'_k corresponding to the received index k to computes his own hash-to-prime τ'_k. Then, he tries to find (q, r) such that $q \in \mathbb{Z}_\mathbb{N}$ and $r \in [l]$ where

$$\left(\prod_{i=1}^{n} \tau'_i\right) \cdot \left(\prod_{j=1}^{m} \tau'_j\right) \cdot {\tau'_k}^{-1} = q \cdot l + r \tag{9}$$

with $\prod_{j=1}^{m} \tau'_j$ is the product of the hashes-to-prime of the new received m-blocks computed by the CSP. Finally, He computes the proof $\pi = g^q$ and sends it concatenated with the witness c'_k to V. When V receives $(\pi \mid\mid c'_k)$, he verifies the proof and accepts if

$$\begin{cases} \tau_k = \tau'_k \leftarrow \mathsf{H_{prime}}(c'_k) \\ r \leftarrow \left(\prod_{i=1}^{n} \tau_i \cdot \left(\prod_{j=1}^{m} \tau'_j\right) \cdot {\tau'_k}^{-1} (\mod l)\right) \\ \pi^l g^r = (\alpha_{t+1})^{\tau_k^{-1}} \end{cases}$$

C. DELETE 1-BLOCK

Moreover, our construction is easily enhanced to support the efficient deletion of one block from the accumulator even without a trapdoor, as shown in Fig. 9. The following steps cover how to delete the j-th block and prove its correct deletion from the remote cloud storage. Fig. 10 illustrates the state of the HPM mapping on the data owner's side and the state of the encrypted blocks $\mathcal{C}$ on the CSP's side before and after the deletion the j-th block. The data owner V:

1) determines the j-th block he wanted to delete.

Preprocessing:
V prepares the data $\mathcal{D}_m$ to add:

1) he splits it into blocks f_j of size l_1; $\mathcal{F}_m \leftarrow \mathcal{D}_m.split(l_1)$,
2) encrypts each block f_j into c_j; $c_j \leftarrow \mathsf{Enc}(\mathsf{vk}, f_j)$ such that $f_j \in \mathcal{F}_m$ and $c_i \in \mathcal{C}$ where $j = \{1, \ldots, m\}$,
3) hashes to prime each block c_j into τ_j; $\tau_j \leftarrow \mathsf{H}_{\mathsf{prime}}(c_j)$ such that $\tau_j \in \mathcal{T}_m$,
4) appends each block τ_j into the HPM mapping; HPM.$save(\tau_j)$ such that $\tau_j \in \mathcal{T}_m$,
5) computes and saves the new accumulated value $\alpha_{t+1} \leftarrow \alpha_t^{\prod_{j=1}^{m} \tau_j}$ over the new set $\mathsf{HPM} \cup \mathcal{T}_m$,
6) chooses $l \xleftarrow{\$} \mathsf{Primes}(\lambda)$ as a challenge with a random index k from HPM.
7) outsources the new encrypted set concatenated with the challenge and the index as $(\mathcal{C}_m \mid\mid l \mid\mid k)$.

ProofGen:
Once P receives $(\mathcal{C}_m \mid\mid l \mid\mid k)$:

1) he finds $c'_k \leftarrow \mathsf{LookupBlock}(\mathcal{C}, k)$ of the challenge element k,
2) computes his own tag; $\tau'_k \leftarrow \mathsf{H}_{\mathsf{prime}}(c'_k)$ of the challenge element k,
3) computes his own version of the tags $\mathcal{T}'_m \leftarrow \mathsf{H}_{\mathsf{prime}}(\mathcal{C}_m)$ of the new set $\mathcal{C}_m$, such that $\mathcal{T}'_m$ is the set of the m-hashes-to-prime of the elements of $\mathcal{C}_m$,
4) inserts $\mathcal{C}_m$ into the correct m positions,
5) tries to find $(q, r) \in \mathbb{Z}_{\mathbb{N}} \times [l]$ such that $\left(\prod_{i=1}^{n} \tau'_i\right) \cdot \left(\prod_{j=1}^{m} \tau'_j\right) \cdot \tau'^{-1}_k = q \cdot l + r$.
6) Then, produces the proof $\pi \leftarrow g^q$, and
7) sends the concatenation $(\pi \mid\mid c'_k)$ of the proof and the witness to V.

ProofVerify:

1) V receives $(\pi \mid\mid c'_k)$, he then verifies and accepts if $\begin{cases} \tau_k = \tau'_k \leftarrow \mathsf{H}_{\mathsf{prime}}(c'_k) \\ r \leftarrow \left(\prod_{i=1}^{n} \tau_i \cdot \left(\prod_{j=1}^{m} \tau'_j\right) \cdot \tau'^{-1}_k (\mod l)\right) \\ \pi^l g^r = (\alpha_{t+1})^{\tau_k^{-1}} \end{cases}$

FIGURE 8. *m*-blocks addition.

2) computes the new accumulated value α_{t+1} over the new set $\mathsf{HPM} \setminus \{\tau_j\}$ such that $\alpha_{t+1} = (\alpha_t)^{\tau_j^{-1}}$, where $\mathsf{HPM} \setminus \{\tau_j\}$ is the old set of hashes-to-prime stored in the HPM mapping with the hash-to-prime of the selected block excluded.
3) sends a sampled random prime $l \xleftarrow{\$} \mathsf{Primes}(\lambda)$ concatenated with the block's index j to P.

Once P receives $(l \mid\mid j)$:

1) he retrieves the encrypted block c'_j of the received index, using the procedure $\mathsf{LookupBlock}(\mathcal{C}, j)$,
2) he computes his own hash-to-prime τ'_j of the $c_j{}'$ and prepares the block $c_j{}'$ to be deleted from the store.
3) Then, P sends $\left(\pi = g^q \mid\mid c'_j\right)$ as non-membership proof of the j-th block, to prove that the deletion was done successfully. We use the same protocol to find $(q, r) \in N \times [l]$ such that (10) holds.

$$\left(\prod_{i=1}^{n} \tau'_i\right) \cdot \tau'^{-1}_j = q \cdot l + r \tag{10}$$

Finally, upon receiving the proof π concatenated with the witness c'_j, the data owner V checks and deletes the hash-to-prime τ_j permanently from the HPM mapping if the follwoing equation (11) holds:

$$\begin{cases} \tau_j = \tau'_j \leftarrow \mathsf{H}_{\mathsf{prime}}(c'_j) \\ r \leftarrow \left(\prod_{\substack{i=1 \\ i \neq j}}^{n} \tau_i (\mod l)\right) \\ \pi^l g^r = \alpha_{t+1} \end{cases} \tag{11}$$

D. DELETE MULTIPLE BLOCKS

Supporting m-blocks deletion is straightforward. It is an extension of the method used for deleting a single block. Instead of just dealing with one position j, it now involves sending a list (vector) of m positions to the *CSP*. Let $\mathcal{C}_d$ be the set of the m ciphertexts and $\mathcal{T}_d$ be the set of the m hashes-to-prime of the elements of $\mathcal{C}_d$ such that $d = (1, \ldots, m)$ is the

Preprocessing:

1) V selects the index j of the block to delete.
2) computes $\alpha_{t+1} \leftarrow \alpha_t^{\tau_j^{-1}}$ over the new set HPM $\setminus \{\tau_j\}$,
3) sends the concatenation $(l \mid\mid j)$ to P such that $l \xleftarrow{\$} \mathsf{Primes}(\lambda)$.

DeleteBlock:

Once P receives $(l \mid\mid j)$:

1) he lookups for the encrypted block j; $\{c_j'\} \leftarrow \mathsf{LookupBlock}(\{c_i'\}_{i=1}^n, j)$,
2) computes he own tag $\tau_j' \leftarrow \mathsf{H_{prime}}(c_j')$.

ProofGen:

1) He tries to find $(q, r) \in \mathbb{Z}_{\mathbb{N}} \times [l]$ such that $\prod_{\substack{i=1\\ i\neq j}}^{n} \tau_i' = q \cdot l + r$,
2) Then, he computes the proof $\pi \leftarrow g^q$, and
3) sends the concatenation $\left(\pi \mid\mid c_j'\right)$ of the proof and the witness c_j' to V.

ProofVerify:

1) V receives $\left(\pi \mid\mid c_j'\right)$, he then verifies and accepts the deletion of the block if the following holds:

$$\begin{cases} \tau_j = \tau_j' \leftarrow \mathsf{H_{prime}}(c_j') \\ r \leftarrow \left(\prod_{\substack{i=1\\ i\neq j}}^{n} \tau_i(\mod l)\right) \\ \pi^l g^r = \alpha_{t+1} \end{cases} \tag{12}$$

FIGURE 9. One block deletion.

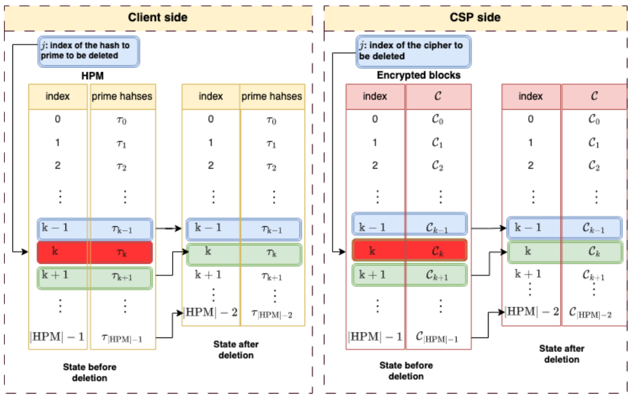


FIGURE 10. The state of the HPM mapping on the client's side and the encrypted blocks on the CSP's side before and after deleting one element.

vector of m positions that V will send to P. Fig. 11 illustrates the process of m-blocks deletion:

1) The data owner V determines a vector $d = (1, \ldots, m)$ of the m-indices he wanted to delete,
2) computes the new accumulated value $\alpha_{t+1} = (\alpha_t)^{\left(\prod_{j=1}^{m} \tau_j\right)^{-1}}$ over the new set HPM $\setminus \mathcal{T}_d$. This set is the old HPM without the set $\mathcal{T}_d$ of the m hashes-to-prime.
3) He randomly select a prime l from $\mathsf{Primes}(\lambda)$, concatenates it with the vector d and an index k randomly selected from the HPM where $k \notin d$. Then, he sends this concatenation $(l \mid\mid d \mid\mid k)$ to the CSP.
4) Once the CSP receives $(l \mid\mid d \mid\mid k)$, he retrieves the m-encrypted blocks $\mathcal{C}_d$ from the data store $\mathcal{C}$, using the vector d as an input to the LookupBlock procedure such that $\mathcal{C}'_d \leftarrow \mathsf{LookupBlock}(\mathcal{C}, d)$,
5) retrieves the kth encrypted block c_k', using the LookupBlock procedure such that $c_k' \leftarrow \mathsf{LookupBlock}(\mathcal{C}, k)$.
6) Then, he tries to find the pair (q, r) such that the equation (13) holds where $(q, r) \in \mathbb{Z}_{\mathbb{N}} \times [l]$.

$$\prod_{\substack{i=1\\ i\notin d}}^{n} \tau_i' = q \cdot l + r \tag{13}$$

7) Finally, he computes the proof $\pi = g^q$, concatenates it with the witness c_k' and sends them to the data owner V.
8) Upon receiving the concatenation $\left(\pi \mid\mid c_k'\right)$, the data owner V verifies the proof π and accepts the deletion

of the m-blocks if the following equation 14 holds:

$$\begin{cases} \tau_k = \tau'_k \leftarrow \mathsf{H}_{\mathsf{prime}}\left(c'_k\right) \\ r \leftarrow \left(\prod_{\substack{i=1 \\ i \notin d}}^{n} \tau_i(\mod l)\right) \\ \pi^l g^r = \alpha_{t+1} \end{cases} \tag{14}$$

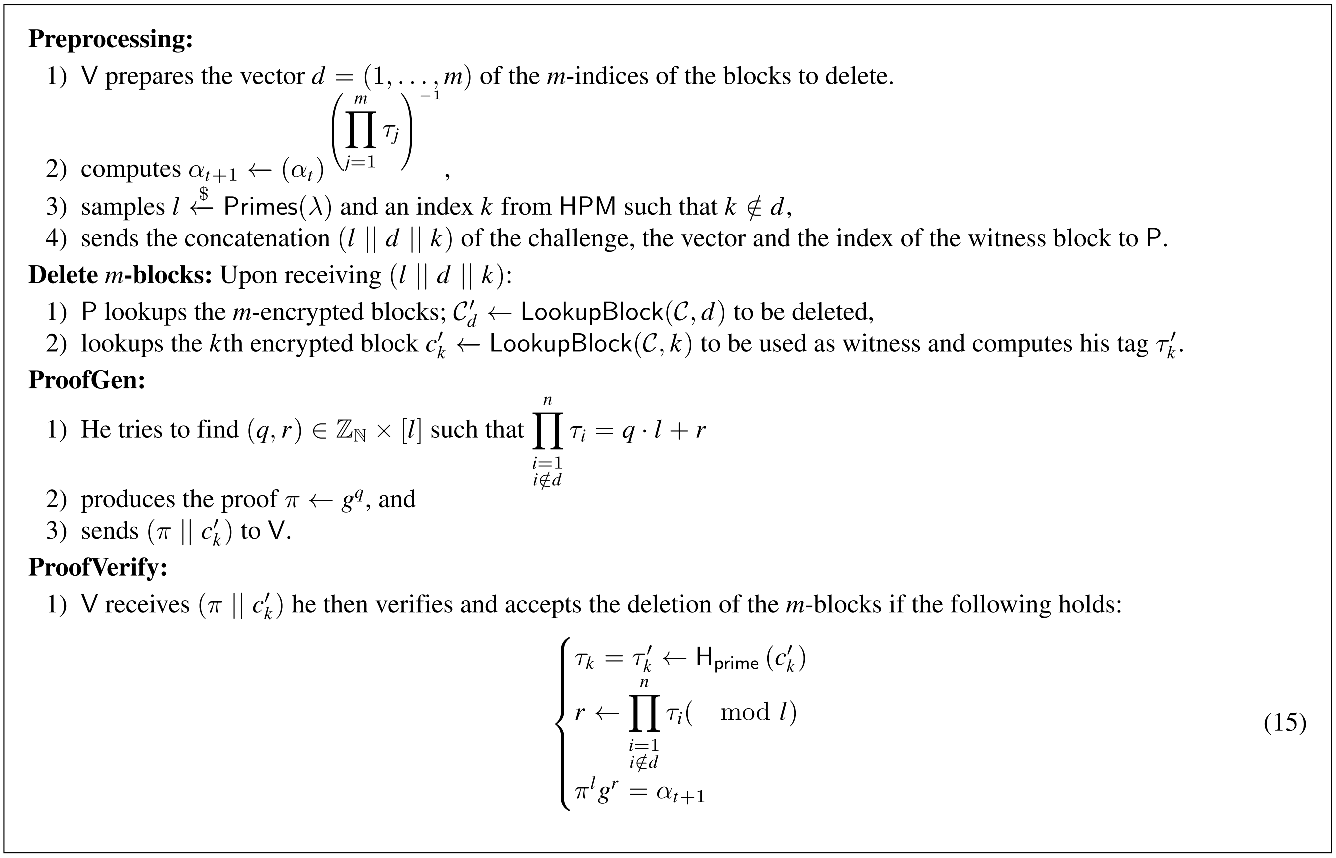
E. UPDATING BLOCKS

Let's say that the data owner wants to update one or more blocks of his outsourced data with some newly created ones. The update operation combines the two previous operations add block(s) and delete(s). Proving the correct update of an element τ_{old_u} against an accumulated value α_t is equivalent to prove that the accumulated value α_t doesn't contain the element τ_{old_u} which corresponds to the delete operation, but contains a new element τ_{new_u} used as a replacement which corresponds to the add operation. Fig. 12 describes how to perform one block update to allow the data owner to substitute the encrypted block c_{old_u} of the index u at the CSP's side with the new block c_{new_u}, and to update it's local HPM mapping with corresponding tag τ_{old_u} with the new one τ_{new_u}. Fig. 13 describes how to perform m-blocks updates. This process requires V to send a vector $d = (1 \ldots m)$ of m-indices to P which maps to the set of blocks which the data owner V wants to update.

VI. CHARACTERISTICS OF POEDDP SCHEME

This section describes the characteristics of our proposed scheme. It follows the cloud's general data integrity verification characteristics as described in [4].

1) **Blockless Verification**: Our scheme removes the need for the data owner to download all the data from the remote data storage to perform integrity verification. The data owner challenges the CSP with a randomly chosen prime number. The CSP generates a proof and sends it back to the data owner, who runs the proof verification.
2) **Unrestricted Challenge Frequency**: Our challenging mechanism does not solely depend on the order of the outsourced data blocks. Our challenge comprises a combination of two sets: a) the set of indices of the blocks being outsourced, and b) the set of odd primes less than 2^{λ} such that λ is the chosen security parameter (i.e. 128, 192, or 256). The set of odd primes less 2^{λ} is considerable in size, and when combined with the set of indices, it provides an effectively unlimited number of challenges.



Preprocessing:
1) V prepares the vector $d = (1, \ldots, m)$ of the m-indices of the blocks to delete.
2) computes $\alpha_{t+1} \leftarrow (\alpha_t)^{\left(\prod_{j=1}^{m} \tau_j\right)^{-1}}$,
3) samples $l \xleftarrow{\$} \mathsf{Primes}(\lambda)$ and an index k from HPM such that $k \notin d$,
4) sends the concatenation $(l \mid\mid d \mid\mid k)$ of the challenge, the vector and the index of the witness block to P.

Delete m-blocks: Upon receiving $(l \mid\mid d \mid\mid k)$:
1) P lookups the m-encrypted blocks; $\mathcal{C}'_d \leftarrow \mathsf{LookupBlock}(\mathcal{C}, d)$ to be deleted,
2) lookups the kth encrypted block $c'_k \leftarrow \mathsf{LookupBlock}(\mathcal{C}, k)$ to be used as witness and computes his tag τ'_k.

ProofGen:
1) He tries to find $(q, r) \in \mathbb{Z}_{\mathbb{N}} \times [l]$ such that $\prod_{\substack{i=1 \\ i \notin d}}^{n} \tau_i = q \cdot l + r$
2) produces the proof $\pi \leftarrow g^q$, and
3) sends $(\pi \mid\mid c'_k)$ to V.

ProofVerify:
1) V receives $(\pi \mid\mid c'_k)$ he then verifies and accepts the deletion of the m-blocks if the following holds:

$$\begin{cases} \tau_k = \tau'_k \leftarrow \mathsf{H}_{\mathsf{prime}}\left(c'_k\right) \\ r \leftarrow \prod_{\substack{i=1 \\ i \notin d}}^{n} \tau_i(\mod l) \\ \pi^l g^r = \alpha_{t+1} \end{cases} \tag{15}$$

FIGURE 11. m-blocks deletion.

Preprocessing:

1) V prepares the new block f_{new_u} of length l_1 that he plans to outsource such that $f_{new_u} \notin \mathcal{F}$,
2) encrypts f_{new_u} such that $c_{new_u} \leftarrow \mathsf{Enc}(\mathsf{vk}, f_{new_u})$
3) hashes to prime the encrypted block c_{new_u} such that $\tau_{new_u} \leftarrow \mathsf{H}_{\mathsf{prime}}(c_{new_u})$
4) computes the new accumulated value over the new set $\mathsf{HPM} \setminus \{\tau_{old_u}\} \cup \{\tau_{new_u}\}$ such that $\alpha_{t+1} \leftarrow \alpha_t^{\tau_{new_u} \cdot \tau_{old_u}^{-1}}$ and saves it locally.
5) He samples a random prime l to be used as a challenge $l \xleftarrow{\$} \mathsf{Primes}(\lambda)$.
6) Then, he outsources the concatenation $(c_{new_u} \mid\mid l \mid\mid u)$ of the new encrypted block c_{new_u}, the challenge l and the index u.

updateBlock:

Once P receives $(c_{new_u} \mid\mid l \mid\mid u)$:

1) he lookups for the encrypted block u, using the *LookupBlock*, $c'_{old_u} \leftarrow \mathsf{LookupBlock}(\mathcal{C}, u)$,
2) replaces the old block c'_{old_u} with the new block c_{new_u} at the index u,
3) computes his own tag $\tau'_{new_u} \leftarrow \mathsf{H}_{\mathsf{prime}}(c_{new_u})$.

ProofGen:

1) He then tries to find $(q, r) \in \mathbb{Z}_{\mathbb{N}} \times [l]$ such that $\displaystyle\prod_{i=1}^{n} \tau_i' \cdot \tau'_{new_u} \cdot \tau'^{-1}_{old_u} = q \cdot l + r$,
2) produces the proof $\pi \leftarrow g^q$, and
3) sends the concatenation $(\pi \mid\mid c'_{old_u})$ of the proof and the witness to V.

ProofVerify:

1) V receives $(\pi \mid\mid c'_{old_u})$, he then verifies and accepts the update if:

$$\begin{cases} \tau_{old_u} = \tau_{old_u}' \leftarrow \mathsf{H}_{\mathsf{prime}}\left(c'_{old_u}\right) \\ r \leftarrow \left(\displaystyle\prod_{i=1}^{n} \tau_i' \cdot \tau_{new_u} \cdot \tau'^{-1}_{old_u} (\mod l)\right) \\ \pi^l g^r = \alpha_{t+1} \end{cases} \tag{16}$$

FIGURE 12. One block update.

3) **Soundness**: Considering the case where the CSP could caches the value $\prod_{i=1}^{n} \tau_i'$ without holding the data. Once the CSP received the challenge, he needs to exclude the hash-to-prime of the requested block j which is τ_j' from the quantity $\prod_{i=1}^{n} \tau_i'$. Reconstructing this value τ_j' from $\prod_{i=1}^{n} \tau_i'$ is unfeasible and considered hard without holding at least holding a mapping of the hashes-to-prime of the deleted data. The fact that the CSP could cache a mapping of the hashes to prime HPM to accelerate the computation won't help him pass the audit. Therefore, he is required to send back the requested jth block to the data owner, which implies that the CSP must hold the full version of the encrypted data. For that reason, our protocol satisfies the soundness characteristic.
4) **Stateless Verification**: Our scheme doesn't rely on previous audits to perform subsequent audits, as the data owner generates a new random prime number l to challenge the CSP for each audit phase. The uniqueness of this prime ensures the uniqueness of the proof π, as finding the pair (q, r) for each l is unique. Storing previous states $(\pi, l_i, q_i, r_i$ on the CSP side will cost additional storage overhead and won't help him bypass the subsequent checks without holding the data.Therefore, our scheme complies with the stateless verification characteristic.
5) **Robustness**: To generate the proof of data possession, the CSP uses all the hashes-to-prime of the encrypted data stored on the HPM except the block being requested to compute the value $\displaystyle\prod_{\substack{i=1 \\ i \neq j}}^{n} \tau_i'$. This value is then used to find the unique pair (q, r) to compute π. Minor tampering with the date will be promptly detected. The auditor receives π concatenated with block c_j' and immediately caught data manipulation during the checking process of $\tau_j' \leftarrow \mathsf{H}_{\mathsf{prime}}(c_j')$ and π against the stored version of τ_j and the accumulated value α_t.
6) **Data Recovery**: Our scheme could be easily enhanced to support data recovery, using any error-correcting codes (i.e., Fast Reed-Solomon Interactive Oracle Proofs of Proximity [48]) during the preprocessing phase to inject the necessary metadata required to do so.

Preprocessing:

1) V prepares the m-blocks f_{new_d} of size $l1$ such that $f_{new_u} \notin \mathcal{F}$ and $u \in d$ where $d = (1 \dots m)$ is the vector of the m-indices V want to update.
2) He encrypts each block f_{new_u} into c_{new_u}; $c_{new_u} \leftarrow \mathsf{Enc}(\mathsf{vk}, f_{new_u})$ such that $u \in d$ to construct the set of ciphertext $\mathcal{C}_{new_d}$,
3) hashes to prime each block c_{new_u} into τ_{new_u}; $\tau_{new_u} \leftarrow \mathsf{H_{prime}}(c_{new_u})$,
4) computes the new accumulated value $\alpha_{t+1} \leftarrow \alpha_t^{(\prod_{u=1}^{m} \tau_{old_u}^{-1}) \cdot (\prod_{u=1}^{m} \tau_{new_u})}$ over the new set of hashes-to-prime $\mathsf{HPM} \leftarrow \mathsf{HPM} \setminus \{\tau_{old_u}\}_{u=1}^{m} \cup \{\tau_{new_u}\}_{u=1}^{m}$ and saves it locally,
5) chooses $l \xleftarrow{\$} \mathsf{Primes}(\lambda)$ as a challenge, and
6) outsources the concatenation $(\mathcal{C}_{new_d} \;||\; l \;||\; d \;||\; k)$ of the ciphertext $\mathcal{C}_{new_d}$, the challenge l, the vector d and the index k to P.

UpdateBlocks: Once P receives $(\mathcal{C}_{new_d} \;||\; l \;||\; d \;||\; k)$:

1) he lookups for each block $u \in d$; ${c_{old_u}}' \leftarrow \mathsf{LookupBlock}(\mathcal{C}, d)$,
2) replaces each block c'_{old_u} with the new block c_{new_u} such that $u \in d$
3) computes his own version of the tags $\mathcal{T}'_{new_d} \leftarrow \mathsf{H_{prime}}(\mathcal{C}_d)$ of the new set $\mathcal{C}_{new_d}$,
4) he lookups for the kth encrypted block; ${c_k}' \leftarrow \mathsf{LookupBlock}(\mathcal{C}, k)$,

ProofGen:

1) P tries to find (q, r) such that $q \in N$ and $r \in [l]$ such that
$$\prod_{i=1}^{n} \tau'_i \cdot \prod_{u \in d} \tau'^{-1}_{old_u} \cdot \prod_{u \in d} \tau'_{new_u} \cdot {\tau_k}'^{-1} = q \cdot l + r$$
2) produces the proof $\pi \leftarrow g^q$,
3) sends the concatenation $(\pi \;||\; c'_k)$ of the proof and the witness to V.

ProofVerify:

1) V receives $(\pi \;||\; c'_k)$ and confirms the update of the m-blocks if:
$$\begin{cases} \tau_k = \tau'_k \leftarrow \mathsf{H_{prime}}\,(c'_k) \\ r \leftarrow \prod_{i=1}^{n} \tau_i \cdot \prod_{u \in d} \tau^{-1}_{old_u} \cdot \prod_{u \in d} \tau_{new_u} \cdot {\tau_k}'^{-1} \\ \pi^l g^r = (\alpha_{t+1})^{\tau_k^{-1}} \end{cases} \tag{17}$$

such that τ_k is the hash-to-prime of the encrypted block c_k compared with the hash-to-prime of the received witness c'_k.

FIGURE 13. ***m*-blocks update.**

7) **Dynamic Data Handling**: Our scheme supports dynamic data operations, making it convenient for real-world applications. It relies on the dynamic mapping of encrypted data blocks with corresponding tags stored in the HPM. Data owners can perform any data operation on the remote cloud storage without downloading the data locally or altering the protocol. Additionally, they can verify the integrity of operations and identify any faulty ones.
8) **Public Auditability**: Our scheme supports public auditing to remove the burden of frequently challenging the CSP. Public auditing involves a Third Party Auditor TPA to perform the blockless verification on behalf of the data owner. It is feasible to delegate this auditing process to a transparent and untrusted TPA that does not require access to private information (i.e., secret keys). This TPA has to operate solely as a computing service on behalf of the data owner. This characteristic cloud be achieved, using any process of public auditing integration [49], [50].
9) **Privacy-Preserving**: Since the data owner encrypts his blocks of data using his secret key vk during the preprocessing, the CSP gains no knowledge of the data. As a result, the client's data remains confidential even in the event of a data breach.
10) **Fairness**: Facing dishonest data owners accusing an honest CSP of data manipulation, it is important to consider the challenges of proving the innocence of the honest CSP. If the CSP lacks the means to provide such proof, especially in a legal setting, the CSP could face a significant financial burden, as disputes of this nature often tend to be biased against the CSP. To this end, our scheme supports fairness to protect the reputation of reliable and honest CSPs. It enables a Law Enforcement Agency LEA to check whether the CSP is innocent or not. Fig. 14 illustrates the interaction between the different actors to achieve fairness.
    1) The LEA requests the necessary public parameters from the data owner.

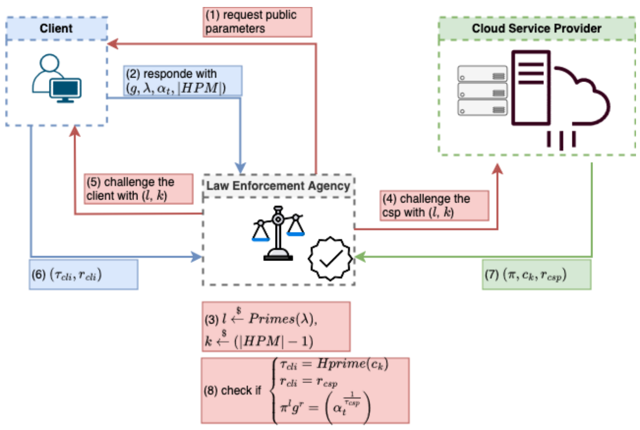

Client
Cloud Service Provider
(1) request public parameters
(2) responde with $(g, \lambda, \alpha_t, |HPM|)$
Law Enforcement Agency
(5) challenge the client with (l, k)
(4) challenge the csp with (l, k)
(6) (τ_{cli}, r_{cli})
(7) (π, c_k, r_{csp})
(3) $l \overset{\$}{\leftarrow} Primes(\lambda)$, $k \overset{\$}{\leftarrow} (|HPM| - 1)$
(8) check if $\begin{cases} \tau_{cli} = Hprime(c_k) \\ r_{cli} = r_{csp} \\ \pi^l g^r = \left(\alpha_t^{\frac{1}{\tau_{csp}}}\right) \end{cases}$

FIGURE 14. Model of checking fairness toward the CSP.

2) The data owner sends g, λ, α_t, $|\mathsf{HPM}|$, the base value, the security parameter, the public binding to the set of elements stored in HPM, and the size of the HPM table, respectively.
3) The LEA samples a prime number l from the set of odd primes in $[0, 2^\lambda]$, and a random index from the HPM table, $k \overset{\$}{\leftarrow} (|\mathsf{HPM}| - 1)$ he challenges the CSP and the data owner with (l, k):
    4) The CSP sends the concatenatation $(\pi \ || \ c_k \ || \ r_{csp})$ of the proof, the jth encrypted block and the remainder such that $\pi = g^q$, and $\prod_{i=1}^{n} \tau_i \cdot \tau_k^{-1} = q \cdot l + r_{csp}$ and $\tau_k = \mathsf{H}_{\mathsf{prime}}(c_k)$ and τ's are the hashes-to-prime of the encrypted blocks c_i's on the CSP side.
    5) the client respond with $(\tau_k \ || \ r_{cli})$ such that $\prod_{i=1}^{n} \tau_i \cdot \tau_k^{-1} = q \cdot l + r_{cli}$ and τ's here are the hashes-to-prime found on the HPM mapping on the client side.
8) The LEA checks if the hash-to-prime of the received c_k from the CSP, equals to the tag τ_k received from the data owner. He also checks if r_{csp} received from the CSP, equals to r_{cli} the following atomic check:

$$\begin{cases} \tau_{cli} = \mathsf{H}_{\mathsf{prime}}(c_k) \\ r_{cli} = r_{csp} \\ \pi^l g^{r_{cli}} = \left(\alpha_t^{\frac{1}{\mathsf{H}_{\mathsf{prime}}(c_k)}}\right) \end{cases} \tag{18}$$

If the equation 18 holds, then the CSP is innocent, else he will be legally accused. The proof of correctness of 18 is as follow:

$$\pi^l g^{r_{cli}} = \left(\alpha_t^{\frac{1}{\mathsf{H}_{\mathsf{prime}}(c_k)}}\right)$$

$$\Leftrightarrow \ < \ def(\pi)^l = (g^q)^l \ , def\,\alpha_t = g^{\prod_{i=1}^{n} \tau_i} \ >$$

$$\Leftrightarrow \ < \ \text{exponents identities} \ >$$

$$g^{ql} g^{r_{cli}} = g^{\prod_{i=1}^{n} \tau_i \cdot \frac{1}{\mathsf{H}_{\mathsf{prime}}(c_k)}}$$

$$\Leftrightarrow \ < \ \text{exponents property} \ >$$

$$ql + r_{cli} = \prod_{i=1}^{n} \tau_i \cdot \frac{1}{\mathsf{H}_{\mathsf{prime}}(c_k)}$$

Q.E.D

This proves that $\prod_{i=1}^{n} \tau_i \cdot \frac{1}{\mathsf{H}_{\mathsf{prime}}(c_k)}$ is uniquely decomposed into $q \cdot l + r_{cli}$. The former quantity, contains $\mathsf{H}_{\mathsf{prime}}(c_k)$, which is the contribution from the CSP side, and $\prod_{i=1}^{n} \tau_i$ which constitute the contribution from the data owner side. The latter quantity also contains q, which is the contribution from the CSP, and r_{cli}, which is the a contribution from the data owner. Additionally, it incorporates a special contribution from the Law Enforcement Agency, represented by the random prime number l.

11) **Deployment Model**: Our PoEDDP scheme could be deployed under the trusted or untrusted setup models as mentioned in II-B5. In the trusted setup model, the trusted party (i.e., the CSP or a trusted authority) will run the GGen algorithm and produce the modulus N. In this case, we required this party to keep the secret key safe. On the other hand, we can remove the trust from a single entity and involve multiple parties in the setup process or use decentralized methods and adopt the untrusted setup model via Multi-Party Computation (MPC) or by a threshold method. In this regard, also, we require that the involved parties can't conspire to forge and then leak the factorization of N to the CSP.

VII. REQUIREMENTS & SECURITY ANALYSIS

The CSP could be considered an active adversary $\mathcal{A}$ with unlimited resources if he decided to act maliciously and potentially mount different vectors to attack the soundness and robustness of our scheme. On one hand, assuming the malicious CSP no longer holds the data, he may attempt to break the hardness of the required assumptions (i.e., Adaptive Root Assumption, Strong-RSA, Division-intractability, Accumulator Security) to forge correct proofs. On the other hand, he could try to break the confidentiality of the client's data by attacking the semantic security of the underlying cipher since he has access to the ciphertext $\mathcal{C}$.

In this section, we defined the security requirements of the PoEDDP scheme. We start with the correct notion of security that our scheme should provide against tampering with the ciphertext. Then, we explore the work deployed by an adversarial CSP claiming to hold the data but does not. Formal proofs are also provided for each security requirement our scheme should meet.

1) IND-CCA REQUIREMENT OF THE CIPHER

Deducing information about the ciphertext, even a minor bit, puts the client's data in a critical situation. To ensure data confidentiality, even from a very powerful CSP, from any adversary $\mathcal{A}$ who could mount active attacks against the ciphertext $\mathcal{C}$. Our scheme uses a Semantically Secure (SS) cipher $\mathsf{Enc}_{poeddp} = (\mathsf{keyGen}, \mathsf{Enc}, \mathsf{Dec})$ that provides Indistinguishability under the Chosen Ciphertext Attack $\mathsf{IND-CCA}$ [51]. The appropriate notion of security, particularly in terms of confidentiality in this context, is established under the $\mathsf{IND-CCA}$ security model. Firstly, when employing Public Key Encryption (PKE) with the PoEDDP protocol, which encrypts data for storage and later consumption by another party. Secondly, when using Symmetric Key Encryption (SKE). To illustrate this notion, a simulation to mesure Enc_{poeddp}'s security is used as follow: Let Enc_{poeddp} be the cipher used in our PoEDDP, defined over the message space $\mathcal{M}$ (block of files f_i), and the ciphertext $\mathcal{C}$, such that keyGen is the key generation algorithm, Enc, and Dec are the encryption, and the decryption algorithms respectively. For a given adversary $\mathcal{A}$. We define the simulation by two experiments ($b = 0$ and $b = 1$). The experiment begins with a key-generation step on the data owner's side to generate a public key pk and verification key vk, considering the PKE case for simplicity, for the security parameter λ, $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{keyGen}(\lambda)$. He then sends the pk to $\mathcal{A}$. We start by allowing $\mathcal{A}$ to make a ξ query to the data owner. Each query is one of the two types:

1) Encryption query: consists of a pair of random messages $(m_{i0}, m_{i1}) \xleftarrow{\$} \mathcal{M}$, where $| m_{i0} |=| m_{i1} |$. The data owner encrypts m_{ib} such that $c_i = \mathsf{Enc}(pk, m_{ib})$, and sends it to $\mathcal{A}$.
2) Decryption query: consists of a randomly chosen ciphertext $\hat{c_j} \xleftarrow{\$} \mathcal{C}^*$ such that $\hat{c_j} \notin \mathcal{C}$ that is not among the responses to the previous encryption queries,

The data owner then, decrypts the received $\hat{c_j}$ such that $\hat{m_j} = \mathsf{Dec}(\mathsf{vk}, \hat{c_j})$, and sends it to $\mathcal{A}$. At the end of the experiment, $\mathcal{A}$ outputs $\hat{b} \in \{0, 1\}$.

Let W_b be the event that $\mathcal{A}$ outputs 1 in the experiment b, and $adv^{\mathsf{IND-CCA}}[\mathcal{A}, \mathsf{Enc}_{poeddp}]$ be the advantage of $\mathcal{A}$ with respect to the cipher Enc_{poeddp} used in PoEDDP to win this security game.

Definition 7: PoEDDP is secure and provides data confidentiality, if for all efficient adversary $\mathcal{A}$, the advantage $adv^{\mathsf{IND-CCA}}[\mathcal{A}, \mathsf{Enc}_{poeddp}]$ of breaking $\mathsf{IND-CCA}$ of the underlying cipher Enc_{poeddp} is negligible.

$$adv^{CCA}[\mathcal{A}, \mathsf{Enc}_{poeddp}] =| Pr[W_0] - Pr[W_1] | \tag{19}$$

2) THE ADAPTIVE ROOT ASSUMPTION REQUIREMENT

Suppose that something went wrong, and the CSP wants to convince the data owner that he holds his data, even if $\alpha_t \neq g^{\frac{\prod \tau_i}{\tau_j}}$ in $\mathbb{G}$. At this stage, we explore the computation power supposed to be deployed by the CSP to achieve his attempt of deducing a correct value to convince the data owner. We demonstrate the necessity for the Adaptive Root assumptions to hold within the group of interest. To illustrate this, we employ a simple root-finding game borrowed from [10], as showcased in Section VII-2. This assumption has been generalized to cover generic groups in [52].

Attack game: The root finding The root finding game is defined over the group $\mathbb{G}$, generated during the setup phase. Let W be the event of an adversary $\mathcal{A}$ of outputting a random element α and then deducing an element u such that $u^l = \alpha$, where l is randomly sampled from $\mathsf{Primes}(\lambda)$. Let $Pr[W]$ be the probability of this event W.

Definition 8: PoEDDP is sound if, for all efficient adversary $\mathcal{A}$, the probability $Pr[W]$ of $\mathcal{A}$ to win the root finding game is negligible.

Proof sketch: if $\mathcal{A}$ fixes a value $\alpha = \alpha_t / g^{\frac{\prod \tau_i}{\tau_j}} \neq 1$ before starting the auditing protocol, he could generate a proof such that $\pi = g^q$. when the challenger generates $l \xleftarrow{\$} \mathsf{Primes}(\lambda)$, and sends it back to $\mathcal{A}$ concatenated with an index j. Given $u = \frac{\pi}{g^q}$, the expression $u^l = \pi^l / g^{ql}$ is derived. We now from the verification condition that $\pi^l = \frac{\alpha_t}{g^r}$ then, $u^l = \left(\frac{\alpha_t}{g^r}\right) / g^{ql}$. Consequently, u^l simplifies to $\alpha = \frac{\alpha_t}{g^{\frac{\prod \tau_i}{\tau_j}}}$. The computational task faced by the adversary $\mathcal{A}$ entails discovering an α, equivalent to computing the l-th root of u, which is known to be a hard problem.

3) THE STRONG-RSA REQUIREMENT

Based on the previous assumption of dealing with a malicious CSP trying to mount a forgery against the accumulated value α_t without holding the ciphertext $\mathcal{C}$ or a version of set of tags $\mathcal{T}$. This adversary $\mathcal{A}$ who on input (N, g) finds both an exponent τ and the τ-th root of g, can forge valid proofs. In doing so, $\mathcal{A}$ solved the Strong-RSA assumption, which is known to be a computationally hard problem.

Given this capability, if $\mathcal{A}$ finds (α, τ) such that $\alpha^\tau = g$ for a given challenge (l, k), the proof then is $\pi = g^q$ such that $1/\tau = q \cdot l + r$. Thus, $\mathcal{A}$ has the ability to create a sequence $\tau_1 \ldots \tau_n$ of tags such that for any challenge k, τ_k divides $\prod \tau_i$, which means that $\mathcal{A}$ may output valid tags $\{\tau_i\}_{i=1}^n \leftarrow \mathsf{H}_{\mathsf{prime}}(\{c_j\}_{i=1}^n)$. Finding valid tags implies that $\mathcal{A}$ finds collisions on $\mathsf{H}_{\mathsf{prime}}$.

Suppose $\mathcal{A}$ finds a collision. Let prove that $\mathcal{A}$ breaks the Strong-RSA problem. Finding a collision means that $\mathcal{A}$ finds a sequence of tags $\tau_1, \ldots, \tau_n$, τ', and a value α'_t such that $(\alpha'_t)^{\tau'} = g^{\prod_{i=1}^{n} \tau_i}$. Using the Extended Euclidean

Algorithm (EEA) and since τ' is prime, $\mathcal{A}$ finds Bèzout coefficients (a, b) such that $a.\prod_{i=1}^{n}\tau_i + b.\tau' = 1$ and puts $\alpha = (\alpha_t')^a.g^b$. Consequently,

$$\begin{aligned}
&\alpha^e = (\alpha_t')^a.g^{be}\\
\Leftrightarrow\ &< \text{exponents properties} >\\
&\alpha^e = (\alpha_t')^{a.e}.g^{b.e}\\
&\alpha^e = ((\alpha_t')^e)^a.g^{b.e}\\
\Leftrightarrow\ &< \text{let } e \text{ be } \tau' \text{ and } (\alpha_t')^{\tau'} = g^{\prod_{i=1}^{n}\tau_i} >\\
&\alpha^e = g^{a.\prod_{i=1}^{n}\tau_i}.g^{b.\tau'}\\
\Leftrightarrow\ &< \text{exponents identity} >\\
&\alpha^e = g^{\prod_{i=1}^{n}\tau_i.a+b.\tau'}\\
\Leftrightarrow\ &< a.\prod_{i=1}^{n}\tau_i + b.\tau' = 1 >\\
&\alpha^e = g\\
&Q.E.D
\end{aligned}$$

Another proof under the generic group model can be found in [52].

4) DIVISION-INTRACTABILITY REQUIREMENT

Finding valid tags, as shown above, won't help $\mathcal{A}$ to convince the data owner because he needs to construct a witness c_k. Hence, another requirement related to $\mathsf{H}_{\mathsf{prime}}$ is needed. This requirement prevents an adversary $\mathcal{A}$ who wants to skip breaking the Strong-RSA problem to adaptively choose a sequence of distinct elements $c_k, c_1, \ldots, c_n$ of the ciphertexts, such that for the challenge $(l \ || \ k)$, $\mathsf{H}_{\mathsf{prime}}(c_k)$ divides $\prod \mathsf{H}_{\mathsf{prime}}(c_i)$. The needed requirement on $\mathsf{H}_{\mathsf{prime}}$ is known as the division-intractability. Constructing such familly of functions is introduced to build a collision-free accumulator and fail-stop signature without trees in [36], and a Hash-and-Sign scheme called GHR after (Gennaro, Halevi, and Rabin) in [53]. Both [36] and [53] show that the security analysis of their scheme could be modeled in the random oracle and the standard model. These types of functions are built from the assumption of collision-resistant of the underlying hash function 2.

Definition 9: The Division-intractability of $\mathsf{H}_{\mathsf{prime}}$ implies that our scheme is existentially unforgeable under the adaptive chosen message attack, assuming the Strong-RSA assumption.

Proof sketch: If $\mathcal{A}$ founds a sequence $c_1, \ldots, c_n$, and c_k such that $\mathsf{H}_{\mathsf{prime}}(c_k)$ divides $\prod \mathsf{H}_{\mathsf{prime}}(c_i)$. Hence, he finds a value t for which $\prod \mathsf{H}_{\mathsf{prime}}(c_i) = t \cdot \mathsf{H}_{\mathsf{prime}}(c_k)$ holds. Then, he easily computes (q_f, r_f) such that $t = \prod \mathsf{H}_{\mathsf{prime}}(c_i) \cdot \mathsf{H}_{\mathsf{prime}}(c_k)^{-1}$ and $t = q_f \cdot l + r_f$, to forge the proof $\pi_f = g^{q_f}$. π_f then is sent with the witness c_k.

5) ACCUMULATOR SECURITY (UNDENIABILITY) REQUIREMENT

Our Accumulator is secure and provides undeniability assuming the Strong-RSA holds in the group $\mathbb{G}$. An adversary $\mathcal{A}$ who wants to crack the security of our accumulator has to forge the tuple $(\alpha_t, \tau_k, \pi_k, \hat{\pi_k})$ such that α_t is the binding commitment, π_k is the proof of inclusion, and $\hat{\pi_k}$ is the proof of exclusion of a given element τ_k. This forgery must holds for *ProofVer* $(\alpha_t, \tau_k, \pi_k)$, and *ProofVer* $(\alpha_t, \tau_k, \hat{\pi_k})$. By definition, as in [52], the proof of exclusion is the pair $\hat{\pi_k} = (a, g^b)$, such that a and b are the Bèzout coefficient between $\prod_{i=1}^{n}\tau_i$ and τ_k where $a \cdot \prod_{i=1}^{n}\tau_i + b \cdot \tau_k = 1$. This proof is verifiable if $\alpha_t^a \cdot g^{b^{\tau_k}} = g \in \mathbb{G}$.

Given the $\mathbb{G} \leftarrow \mathsf{GGen}(\lambda)$ and a base value $g \xleftarrow{\$} \mathbb{G}$ to the adversary $\mathcal{A}$, if $\mathcal{A}$ successfully outputs the tuple $(\alpha_t, \tau_k, \pi_k, \hat{\pi_k})$, he can put $g^b \cdot (\pi_k)^a \in \mathbb{G}$ to be the x-th root of g. Since By definition, $\alpha_t^a \cdot g^{b^{\tau_k}} = g \in \mathbb{G}$ and the forge gives $\left(g^b \cdot (\pi_k)^a\right)^{\tau_k} = g \in \mathbb{G}$, thus we get $\alpha_t^a \cdot g^{b \cdot \tau_k} = \left(g^b \cdot (\pi_k)^a\right)^{\tau_k} \in \mathbb{G}$. The right hand side is then simplied into $g^{b \cdot \tau_k} \cdot \alpha_t^a \in \mathbb{G}$, using the law of exponentiation twice and knowing that $\alpha_t = \pi_k^{\tau_k}$. Thus, we have $\alpha_t^a \cdot g^{b^{\tau_k}} = g^{b \cdot \tau_k} \cdot \alpha_t^a \in \mathbb{G}$ which contradicts that the Strong-RSA assumption holds in $\mathbb{G}$ and implies that the accumulator is undeniable.

VIII. PERFORMANCE ANALYSIS

In this section, we conduct a performance analysis of our PoEDDP scheme compared to the scheme proposed by Khadr et al. [9]. Both schemes share similar characteristics, including being deterministic RSA-based Accumulators.

First, We quantify the computation, communication, and storage costs incurred by the participating parties, using the similar notation used in [9] as shown in Table 1, such that 1) ($s1 = |\tau| = |l| = |r| = 128$-bits), be the size of tags, the prime challenge, and the remainder of the division modulo l respectively. 2) ($s2 = |N| = |q| = 3072$-bits), be the size of the RSA-moduli, and the quotient of the division modulo l respectively. 3) ($s3 = |\phi(N)| = 3071$-bits), be the of the Euler's phi function, 4) ($s4 = |b_i''| = 3327$-bits),be the size of the generated block in Khadr et al.s scheme, These cost metrics provide a basis to benchmark our scheme against Khadr et al. scheme's [9] and compare different optimizations to improve our PoEDDP scheme. The selected metrics also help us assess how well our scheme fits real-world scenarios with resource constraints.

Afterward, we compare the performance of a prototype of our scheme to the previously mentioned scheme, using Z-score normalization. It's important to emphasize that the use of this normalization distort the results and obscure certain performance details. To provide a more accurate assessment, we also present the unnormalized results for both schemes.

A. COMPUTATION COST

Our scheme operates through three phases: setup, pre-processing, and auditing. The setup phase is a one-time

TABLE 1. Notations of operation costs.

Symbol	Operation cost
T_{muls1}	modular integer multiplication with s1 bits operands
T_{muls4}	modular integer multiplication with s4 bits operands
T_{exps2}	modular exponentiation s2 bit exponent
T_{exps4}	modular exponentiation s4 bit exponent
T_{div}	modular division
T_{inv}	modular inverse
T_{rand}	random number sampling
T_{Rprime}	random prime sampling
T_{hash}	hashing
$T_{\mathsf{H}_{\mathsf{prime}}}$	Hash-to-Prime function
T_{Enc}	encryption
T_{lookup}	lookup operation
T_{setup}	setup cost
T_{pre}	Pre-processing cost
T_{aud}	audit cost
T_{chal}	challenge cost
T_{pGen}	proof generation
T_{pVer}	proof verification
T_{comp}	comparison operation

operation following a setup model of choice II-B5. For simplicity, we will keep it on the client side assuming a trused setup. This phase takes $T_{setup} = 2T_{Rprime} + T_{muls2} + T_{rand}$. The pre-processing phase depends on the size of file $\mathcal{F}$ and lets n be the number of data blocks. This phase takes $T_{pre} = nT_{\mathsf{Enc}} + nT_{\mathsf{H}_{\mathsf{prime}}} + nT_{muls1} + T_{exps2}$. The auditing phase comprises three steps; challenge, proof Generation, and proof Verification; each introducing different overhead levels. This final phase takes $T_{aud} = T_{chal} + T_{pGen} + T_{pVer}$, where the challenge step introduces a random selection of prime number $T_{chal} = T_{Rprime}$, the proof generation takes $T_{pGen} = T_{lookup} + (n-1)T_{muls1} + T_{div} + T_{exps2}$, and the proof verification takes $T_{pVer} = T_{\mathsf{H}_{\mathsf{prime}}} + (n-1)T_{muls1} + T_{div} + T_{inv} + 3T_{exps1} +$.

As illustrated above, we have enumerated every computation task, including minor ones. However, for clarity and conciseness in the performance comparison regarding computation cost between our scheme and [9], we omitted less resource-intensive operations as it makes sense to highlight those that require significant computational resources or are time-consuming, as these are often the bottlenecks in the scheme's performance. Also we merged the setup phase with the pre-processing phase and called it the setup phase. Thus, the final computation cost is summarized in Table 2.

B. COMMUNICATION COST

In principle, The data owner has to send (N, g, l, j), in addition to the size of data sent to the CSP. We apply the Fiat Shamir transformation [54] to reduce the challenge size into $|j|$, then the CSP responds with $\left(\pi \ || \ c'_j\right)$. Thus our scheme introduces a communication cost of $|N| + |g| + |j| + |\pi| + c'_j$.

C. STORAGE COST

The CSP has to store the $|N| + |g| + |\{ci\}|$, the modulus, a group element, and the outsourced data, respectively. Conversely, the data owner stores $|N| + |g| + |\mathsf{HPM}| + |\mathsf{vk}| + |\alpha_t|$, the modulus, a group element, the Hash-to-Prime Mapping HPM, and the digest, respectively. Table 2 shows a performance comparison between our PoEDDP scheme and khadr's scheme [9].

D. NUMERICAL SIMULATIONS

We implement a prototype version of our scheme, as published on GitHub [55], using python-3.11.4 and pycryptodomex library [56]. We opted to use Python for our cryptographic simulations due to its versatility and ease of use. This made it an ideal choice to construct a fast Proof of Concept that could be easily tested across multiple platforms, ensuring a broad scope for evaluation. Pycryptodomex is a widely used and self-contained Python package of low-level cryptographic primitives. It offers the advantage of being a well-maintained, comprehensive library that meets the stringent requirements of cryptographic operations while ensuring ease of integration into our Python-based simulations.

We construct the RSA-Accumulators for both schemes, using a 3072-bit RSA modulus to ensure 128-bit security. We encrypt random blocks of 348 bytes, using AES-128-GCM to get 416 bytes for each cipher-text, including the tag and nonce, and hashing was done, using SHA256. AES-128-GCM and SHA256 are imported from the Pycryptodomex library. The set of blocks consists of increments of 10k blocks, each with a size of 416 bytes. The block count ranges from 10k to 100k blocks. This upper limit of 100k blocks, equivalent to approximately 42 megabytes, was chosen to ensure that the computation time for Khadr et al.'s scheme [9] remains within an acceptable range (3 hrs 29 min 27.98 sec). It is worth noting that the 384-byte block selection is a deliberate choice made with a specific purpose. We aim to conduct a performance comparison that directly competes with Khadr et al.'s scheme in their realm and under their optimized implementation. This choice enables us to rigorously evaluate our scheme against theirs and showcase its capabilities under the conditions most relevant to them. We believe that this approach provides valuable insights and a fair basis for comparison, highlighting the strengths and advantages of our scheme.

We conducted the numerical simulations without any optimizations or alterations to the number of hardware threads and parallelization. The simulations were performed ten (10) times on a MacBook Air M1 with MacOS Ventura 13.5.2 installed and updated. The MacBook Air M1 features an 8-core CPU with 4 performance cores, 4 efficiency cores, a 7-core GPU, a 16-core Neural Engine, and 8 GB of RAM. The average computation cost for each phase was then calculated. In the results illustrated in Figures Fig. 15, Fig. 16, Fig. 17, and Fig. 18, we depict the number of encrypted blocks processed on the x-axis, while the computation time in seconds for various phases is represented on the y-axis.

Fig. 15 shows a full comparison of our proposed scheme and Khadr et al.'s scheme [9]. It illustrates the normalized

TABLE 2. Performance comparison between our PoEDDP scheme and Khadr et al.'s scheme.

Costs		Our PoEDDP Scheme	Khadr et al.'s scheme [9][a]
Computation	Setup	$nT_{\mathsf{Enc}} + nT_{\mathsf{H}_{\mathsf{prime}}} + nT_{muls1} + T_{exps2}$	$nT_{\mathsf{Enc}} + nT_{hash} + nT_{muls4} + T_{exps2}$
	Challenge Gen	T_{rand}	T_{rand}
	Proof Gen	$(n-1)T_{muls1} + T_{exps2}$	$(n-1)T_{muls4} + T_{exps4}$
	Proof Verification	$3T_{exps1}$	T_{exps4}
Communication		$\|j\| + 2\|N\| + \|c_j\|$	$\|j\| + \|N\| + \|\phi(N)\| + \|w_j\| + \|b_j^{''}\|$
Storage	Data owner	$n\|l_3\| + \|N\| + \|g\| + \|\alpha_t\| + \|\mathsf{pk}\| + \|\mathsf{vk}\|$	$n\|l_h\| + \|N\| + \|g\| + \|\phi(N)\| + \|acc_t\| + \|K_E\| + \|K_H\|$
	CSP	$\|N\| + \|g\| + \|\{c_i\}_{i=1}^{n}\|$	$\|N\| + \|g\| + \|\phi(N)\| + \|\{b_i^{''}\}_{i=1}^{n}\|$

[a]: The notation used in [9] are; K_E encryption key, K_H hashing key, $\phi(N)$ Euler's phi function, acc_t accumulated value (digest), $b_i^{''}$ generated block, l_h length of the hashed block, w_j j-th witness

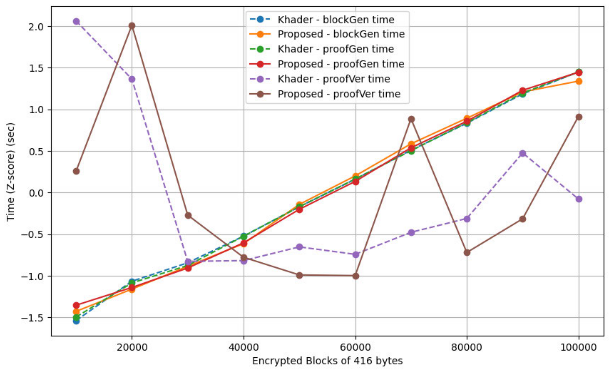


FIGURE 15. Performance of the proposed and Khadr et al.'s scheme on different set of blocks of 416 bytes with time normalized, using Z-score.

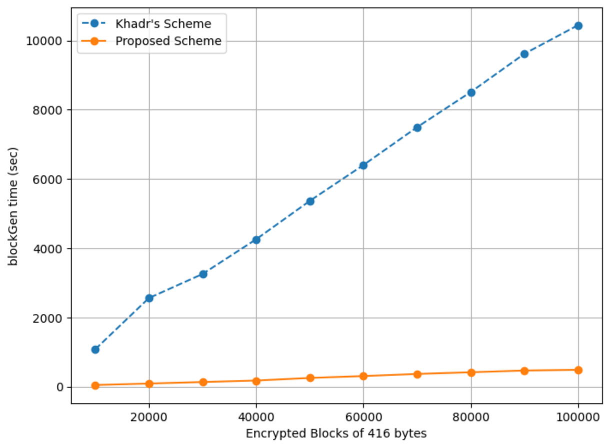


FIGURE 16. Block Generation time comparison of the proposed scheme and Khadr et al.'s scheme on different set of blocks of 416 bytes with a non normalized time.

costs of block generation, proof generation, and proof verification, using Z-scores. It shows that, in both schemes, the time required for block generation and proof generation exhibits a linear relation in function of the number of processed blocks. On the other hand, verification times in both schemes exhibit a significant rate of change from higher values when processing 10,000 and 20,000 blocks to lower values around 30,000 blocks. We can consider the proof verification as independent of the number of blocks.

Fig. 16 illustrates the computation cost in both schemes during the block generation phase without Z-score normalization. This phase is the most critical and time-consuming in both cases. It encompasses several tasks, including dividing large files into blocks, encrypting these blocks, generating signatures,applying transformations when necessary (e.g., left shifts by 1 as in [9] and hash-to-primes in our scheme), and computing the binding value through large integers multiplication and exponentiation. It is worth noting that the large integer multiplication and the exponentiation represent the bottleneck in this process, as they require intensive computational resources. In both schemes, this costs is linear in the number of the processed blocks, but the rate of change in Khadr et al.'s scheme is faster compared to ours.

Fig. 17 illustrates the computation cost in both schemes during the proof generation phase without Z-score normalization. It involves large integer multiplication and exponentiation. The former is required, in our scheme, to deterministically accumulate all tags into one value to be used in the Euclidean division. The latter is required by [9] to compute the witness. One can deduce that performing large integer multiplication has less computation overhead compared to the exponentiation. This cost is also linear in the number of elements processed and the rate of change in Khadr et al.'s scheme is faster compared to ours.

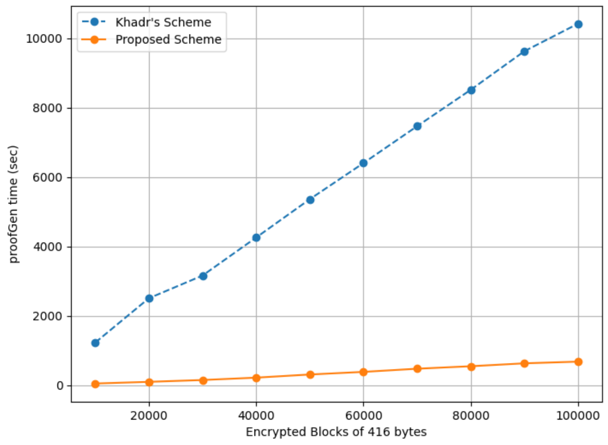


FIGURE 17. Proof Generation time comparison of the proposed scheme and Khadr et al.'s scheme on different set of blocks of 416 bytes with a non normalized time.

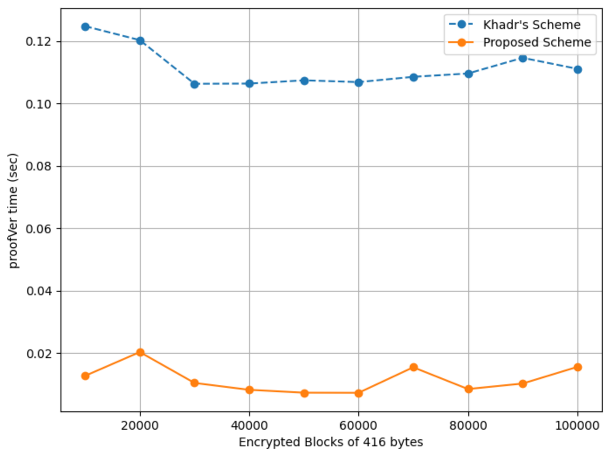


FIGURE 18. **Proof Verification time comparison of the proposed scheme and Khadr et al.'s scheme on different set of blocks of 416 bytes with a non normalized time.**

TABLE 3. **The average scaling factors for 'blockGen time', 'proofGen time', and 'proofVer time' between the Khadr scheme and our proposed scheme.**

Phase	Avg. Scaling factor
blockGen time	24.206880
proofGen time	24.511442
proofVer time	14.343414

The last Fig. 18 illustrates the computation cost in both schemes during the proof verification phase without Z-score normalization. It is apparent that this phase is faster in both cases compared to other phases and tends to exhibit quasi-constant time behavior. Additionally, the figure demonstrates that our scheme is faster compared to Khadr et al's scheme by an order of magnitude.

Figures 16, 17, and 18, provide clear evidence of our scheme's superior performance compared to Khadr et al.' scheme across all three phases. The summarized results in Table 3 highlight the significant speed advantage of our scheme. Specifically, it surpasses Khadr et al.' scheme by approximately a factor of 24 in the block generation and proof generation phases and by approximately a factor of 14 in the verification phase. These findings affirm the substantial performance gains achieved by our scheme.

IX. DISCUSSION

Initially, it appears that the comparison, using Z-score normalization, as depicted in Figure 15, might suggest that our scheme offers no advantage over Khadr et al.'s scheme and that both schemes are useless for small files with fewer than 30k blocks. This is because the time required for proof verification is considerably longer than for block generation and proof generation. While the initial impression may lack an advantage for our scheme over Khadr et al.'s scheme, the simulations without Z-score normalization reveal significant distortion to the results. Thus, when the comparisons were performed without Z-score normalization as illustrated in figures Fig. 16, 17, and 18, our scheme proved substantially faster, outperforming Khadr's by factors of 24x in block generation and in proof generation, and 12x in the proof verification, as evidenced in Table 3. In Khadr et al.'s scheme, each block undergoes encryption, hashing, and concatenation with the corresponding tag. Following this, a left shift by one position is applied to create a final block, which serves as non-coprime representatives. As per the simulation parameters detailed in Section VIII-D, this block has a size of 3327 bits and is used in the subsequent multiplication and exponentiation steps. Performing large integer multiplication, each approximately 3327 bits in size, modulo $\phi(N)$ of size approximately 3071 bits, represents a significant computational bottleneck in this task. The impact is visible, and we saw significant performance degradation when dealing with sets of significant amounts of blocks, especially during the block generation and proof generation phases Fig. 16, and 17 respectively. In contrast, our proposed scheme encrypts each block, concatenates it with the corresponding tag, and then applies a hash-to-prime transformation to create a final odd prime of size 128 bits used for multiplications. While this step incurs a hashing process, which requires additional processing time, we gain in terms of performance and efficiency during subsequent multiplication. As a result, the multiplication of 128-bit integers is unlikely to present a significant bottleneck in our scheme for medium to high numbers of blocks during block generation and proof generation.

In the performance evaluation above, Khadr's scheme and our proposed scheme were executed, using specified settings VIII-D, employing AES-128-GCM and SHA256 on a MacBook Air equipped with an M1 (arm) processor. It is crucial to note that modifying AES encryption modes and parameters from AES-128-GCM can significantly impact the security and performance of both schemes. While this modification does not affect the block size, given the fixed output size of AES, it introduces an additional computation overhead during the encryption process in the block generation (blockGen) phase. Switching to alternative ciphers, which produce outputs exceeding 128 bits, exclusively affects Khadr's scheme. This alteration extends the length of the final block, consequently influencing the modular multiplication and modular exponentiation phases when dealing with the larger block size. Furthermore, replacing SHA256 with another cryptographic hash function (CRH) like SHA3 [57] or Keccak [58] has implications for both schemes. In Khadr's scheme, this substitution will make the size of the final block larger ($>$ 3327 bits). Thus, in addition to the computation overhead on the hashing process (tag creation), it will introduce more overhead to the modular multiplication and exponentiation bottleneck. Consequently, the scheme will be impractical when dealing with low-power devices. Conversely, our scheme is impacted solely by the additional overhead of the hashing process; however, other critical phases, such as modular multiplication and exponentiation, remain unaffected due to the fixed size of the final block set at 128 bits during the hash-to-prime process. Another drawback of both schemes is the usage of RSA; thus, they inherited RSA limitations that we do not tackle in this work. For comprehensive benchmarking, future considerations may involve exploring additional parameters, particularly those relevant to post-quantum cryptography, such as Crystals and Kyber [59].

X. CONCLUSION

We introduce PoEDDP, a member of the Proof of Data Possession (PDP) family is implemented, using a variant of RSA-Accumulators based on Proof of Exponentiation PoE. The scheme allows data owners to continuously audit the integrity and availability of the outsourced data. The audit guarantees that the CSP deterministicly uses the stored data to compute the proof of data possession. This proof is then verified against a single accumulated value α_t computed from small footprints of tags stored in HPM. Additionally, the scheme offers dynamic capabilities to execute various operations on the outsourced data, including additions, updates, and deletions within the $\mathsf{CSP}'s$ end.

Comparatively, our scheme significantly outperforms Khadr's scheme [9]. To provide a practical example, our proof generation phase, conducted on a dataset of 100k blocks, each consisting of 416 bytes, requires a mere (8 min 4 sec). In contrast, Khadr's scheme takes (2 hrs 54 min 4 sec) under the same conditions. These results are obtained with naive Proof of Concept (PoC) without specialized libraries or optimizations tailored for large integer multiplication. Thus, it excels in computational efficiency across all phases, even when processing extensive datasets.

Our proposed scheme offers several advantages compared to Khadr et al.; however, it is worth acknowledging the current limitations. Both schemes are built on top of RSA-groups. Thus, they rely on the intractability problem of the RSA modulus. On one hand, a sufficiently large modulus, such as 3072 bits for 128-bit security, is essential but inevitably leads to decreased performance, especially when handling larger moduli. On the other hand, the intractability of the RSA modulus is facing a significant threat in the post-quantum era, as done for the widely used RSA-2048 by Yan et al. in [60], using the classical lattice reduction with a quantum approximate optimization algorithm. Additionally, both schemes primarily target data possession within remote servers, focusing on ensuring data integrity and possession. However, they fall short in addressing a vital aspect of the proof of data retrievability, which represents a significant limitation in both schemes.

Thus, as researchers, we may consider whether we can develop a scheme capable of delegating the most resource-intensive tasks (e.g., block generation) to the CSP while ensuring both the proof of correct computation and the proof of data retrievability, all within the context of an untrusted setup and taking into account the quantum computing threat.

REFERENCES

[1] *Public Cloud Infrastructure Spending Worldwide 2015–2026*, Statista, Hamburg, Germany, 2017.

[2] *Cloud Computing Market Size & Share | Growth Analysis*, Fortunebusinessinsights, Maharashtra, India, 2030.

[3] *Number of Connected IoT Devices Growing 9% to 12.3 Billion Globally, Cellular IoT Now Surpassing 2 Billion*, IoT.Business.News, Paris, France, Sep. 2021.

[4] F. Zafar, A. Khan, S. U. R. Malik, M. Ahmed, A. Anjum, M. I. Khan, N. Javed, M. Alam, and F. Jamil, "A survey of cloud computing data integrity schemes: Design challenges, taxonomy and future trends," *Comput. Secur.*, vol. 65, pp. 29–49, Mar. 2017.

[5] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology—CRYPTO '87*, Santa Barbara, CA, USA. Cham, Switzerland: Springer, Aug. 1987, pp. 369–378.

[6] A. Ozdemir, R. S. Wahby, B. Whitehat, and D. Boneh, "Scaling verifiable computation using efficient set accumulators," in *Proc. 29th USENIX Conf. Secur. Symp.*, 2020, Paper 117. [Online]. Available: https://dl.acm.org/doi/proceedings/10.5555/3489212

[7] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Proc. Annu. Cryptol.-CRYPTO Annu. Int. Cryptol. Conf.* Santa Barbara, CA, USA: Springer, Aug. 2002, pp. 61–76.

[8] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to IOPs and stateless blockchains," in *Advances in Cryptology—CRYPTO 2019*, vol. 11692. Cham, Switzerland: Springer, 2019, pp. 561–586.

[9] W. I. Khedr, H. M. Khater, and E. R. Mohamed, "Cryptographic accumulator-based scheme for critical data integrity verification in cloud storage," *IEEE Access*, vol. 7, pp. 65635–65651, 2019.

[10] B. Wesolowski, "Efficient verifiable delay functions," in *Advances in Cryptology—EUROCRYPT 2019*. Cham, Switzerland: Springer, 2019, pp. 379–407.

[11] J. Benaloh and M. de Mare, "One-way accumulators: A decentralized alternative to digital signatures," in *Advances in Cryptology—EUROCRYPT '93*, G. Goos, J. Hartmanis, and T. Helleseth, Eds. Berlin, Germany: Springer, 1994, vol. 765, pp. 274–285.

[12] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proc. 14th Eur. Symp. Res. Comput. Secur.*, Saint-Malo, France, Sep. 2009, pp. 355–370.

[13] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proc. 18th ACM Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Oct. 2011, pp. 491–500.

[14] R. Du, L. Deng, J. Chen, K. He, and M. Zheng, "Proofs of ownership and retrievability in cloud storage," in *Proc. IEEE 13th Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Sep. 2014, pp. 328–335.

[15] C. Liu, R. Ranjan, C. Yang, X. Zhang, L. Wang, and J. Chen, "MuR-DPA: top-down levelled multi-replica Merkle hash tree based secure public auditing for dynamic big data storage on cloud," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2609–2622, Sep. 2015.

[16] L. Xue, J. Ni, Y. Li, and J. Shen, "Provable data transfer from provable data possession and deletion in cloud storage," *Comput. Standards Interface*, vol. 54, pp. 46–54, Nov. 2017.

[17] W. Guo, H. Zhang, S. Qin, F. Gao, Z. Jin, W. Li, and Q. Wen, "Outsourced dynamic provable data possession with batch update for secure cloud storage," *Future Gener. Comput. Syst.*, vol. 95, pp. 309–322, Jun. 2019.

[18] K. He, J. Chen, Q. Yuan, S. Ji, D. He, and R. Du, "Dynamic group-oriented provable data possession in the cloud," *IEEE Trans. Depend. Secure Comput.*, vol. 18, no. 3, pp. 1394–1408, May 2021.

[19] J. Mao, Y. Zhang, P. Li, T. Li, Q. Wu, and J. Liu, "A position-aware Merkle tree for dynamic cloud data integrity verification," *Soft Comput.*, vol. 21, no. 8, pp. 2151–2164, Apr. 2017.

[20] R. Mishra, D. Ramesh, and D. R. Edla, "Dynamic large branching hash tree based secure and efficient dynamic auditing protocol for cloud environment," *Cluster Comput.*, vol. 24, no. 2, pp. 1361–1379, Oct. 2020.

[21] J. Li, N. Li, and R. Xue, "Universal accumulators with efficient nonmembership proofs," in *Proc. ACNS*. Berlin, Germany: Springer, 2007, pp. 253–269.

[22] T. Sander, "Efficient accumulators without trapdoor extended abstract," in *Information and Communication Security*, vol. 1726, G. Goos, J. Hartmanis, J. van Leeuwen, V. Varadharajan, and Y. Mu, Eds. Berlin, Germany: Springer, 1999, pp. 252–262.

[23] K. Nyberg, ''Fast accumulated hashing,'' in *Fast Software Encryption*, vol. 1039, Cambridge, U.K. Cham, Switzerland: Springer, Feb. 1996, pp. 83–87.
[24] H. Lipmaa, ''Secure accumulators from euclidean rings without trusted setup,'' in *Applied Cryptography and Network Security*, vol. 7341, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. P. Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, F. Bao, P. Samarati, and J. Zhou, Eds. Berlin, Germany: Springer, 2012, pp. 224–240.
[25] E. Ghosh, O. Ohrimenko, D. Papadopoulos, R. Tamassia, and N. Triandopoulos, ''Zero-knowledge accumulators and set algebra,'' in *Proc. ASIACRYPT*. Springer, 2016, pp. 67–100.
[26] B. Libert, S. Ling, K. Nguyen, and H. Wang, ''Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors,'' in *Proc. EUROCRYPT*, 2016, pp. 1–31.
[27] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, ''Provable data possession at untrusted stores,'' in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, Oct. 2007, pp. 598–609.
[28] G. Caronni and M. Waldvogel, ''Establishing trust in distributed storage providers,'' in *Proc. 3rd Int. Conf. Peer Peer Comput. (P2P)*, Sep. 2003, p. 128.
[29] Y. Deswarte, J.-J. Quisquater, and A. Saïdane, ''Remote integrity checking,'' in *Integrity and Internal Control in Information Systems VI*, vol. 140, S. Jajodia and L. Strous, Eds. Boston, MA, USA: Kluwer, 2004, pp. 1–11.
[30] D. L. G. Filho and P. S. L. M. Barreto, ''Demonstrating data possession and uncheatable data transfer,'' Cryptol. ePrint Arch., Tech. Rep. 2006/150, 2006. [Online]. Available: https://eprint.iacr.org/2006/150
[31] F. Sebe, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J.-J. Quisquater, ''Efficient remote data possession checking in critical information infrastructures,'' *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 8, pp. 1034–1038, Aug. 2008.
[32] Z. Hao, S. Zhong, and N. Yu, ''A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability,'' *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 9, pp. 1432–1437, Sep. 2011.
[33] M. Yi, J. Wei, and L. Song, ''Efficient integrity verification of replicated data in cloud computing system,'' *Comput. Secur.*, vol. 65, pp. 202–212, Mar. 2017.
[34] Y. Ren, J. Qi, Y. Liu, J. Wang, and G.-J. Kim, ''Integrity verification mechanism of sensor data based on bilinear map accumulator,'' *ACM Trans. Internet Technol.*, vol. 21, no. 1, pp. 1–19, Feb. 2021.
[35] P. Camacho, A. Hevia, M. Kiwi, and R. Opazo, ''Strong accumulators from collision-resistant hashing,'' in *Proc. Int., Conf. Inf. Secur.* Cham, Switzerland: Springer, 2008, pp. 471–486.
[36] N. Bari and B. Pfitzmann, ''Collision-free accumulators and fail-stop signature schemes without trees,'' in *Advances in Cryptology—EUROCRYPT '97*, vol. 1233, Konstanz, Germany. Cham, Switzerland: Springer, May 1997, pp. 480–494.
[37] L. Nguyen, ''Accumulators from bilinear pairings and applications,'' in *Topics Cryptology–CT-RSA*, vol. 3376, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, and A. Menezes, Eds. Berlin, Germany: Springer, 2005, pp. 275–292, doi: 10.1007/978-3-540-30574-3_19.
[38] I. Damgård and N. Triandopoulos, ''Supporting non-membership proofs with bilinear-map accumulators,'' Cryptol. ePrint Arch., Tech. Paper 2008/538, 2008. [Online]. Available: https://eprint.iacr.org/2008/538
[39] J. Camenisch, M. Kohlweiss, and C. Soriente, ''An accumulator based on bilinear maps and efficient revocation for anonymous credentials,'' in *Public Key Cryptography PKC*, vol. 5443, Irvine, CA, USA. Cham, Switzerland: Springer, Mar. 2009, pp. 481–500.
[40] D. Boneh, B. Bünz, and B. Fisch, ''A survey of two verifiable delay functions,'' Cryptol. ePrint Arch., Tech. Paper 2018/712, 2018. [Online]. Available: https://eprint.iacr.org/2018/712
[41] C. Hoffmann, P. Hubácek, C. Kamath, K. Klein, and K. Pietrzak, ''Practical statistically-sound proofs of exponentiation in any group,'' in *Advances in Cryptology—CRYPTO*, Y. Dodis and T. Shrimpton, Eds. Cham, Switzerland: Springer, 2022, pp. 370–399. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-15979-4_13
[42] F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin, and S. Yakoubov, ''Accumulators with applications to anonymity-preserving revocation,'' Tech. Rep. 043, 2017.
[43] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza, ''Secure sampling of public parameters for succinct zero knowledge proofs,'' in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 287–304, doi: 10.1109/SP.2015.25.
[44] S. Bowe, A. Gabizon, and I. Miers, ''Scalable multi-party computation for zk-snark parameters in the random beacon model,'' Cryptol. ePrint Arch., Tech. Rep. 2017/1050, 2017.
[45] M. Kohlweiss, M. Maller, J. Siim, and M. Volkhov, ''Snarky ceremonies,'' *Advances in Cryptology—ASIACRYPT*, M. Tibouchi and H. Wang, Eds. Cham, Switzerland: Springer, 2021, pp. 98–127. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-92078-4_4
[46] V. Nikolaenko, S. Ragsdale, J. Bonneau, and D. Boneh, ''Powers-of-tau to the people: Decentralizing setup ceremonies,'' Cryptol. ePrint Arch., Tech. Paper 2022/1592, 2022. [Online]. Available: https://eprint.iacr.org/2022/1592
[47] S. Das, Z. Xiang, and L. Ren, ''Powers of Tau in asynchrony,'' Cryptol. ePrint Archive, Tech. Paper 2022/1683, 2022. [Online]. Available: https://eprint.iacr.org/2022/1683
[48] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, ''Fast Reed–Solomon interactive oracle proofs of proximity,'' in *Proc. 45th Int. Colloq. Automata, Lang., Program. (ICALP)*, 2018. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2018.14
[49] G. Ateniese, S. Kamara, and J. Katz, ''Proofs of storage from homomorphic identification protocols,'' in *Advances in Cryptology—ASIACRYPT 2009*, Tokyo, Japan. Cham, Switzerland: Springer, Dec. 2009, pp. 319–333.
[50] W. Luo and G. Bai, ''Ensuring the data integrity in cloud data storage,'' in *Proc. IEEE Int. Conf. Cloud Comput. Intell. Syst.*, Sep. 2011, pp. 240–243.
[51] S. Goldwasser and S. Micali, ''Probabilistic encryption,'' *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.
[52] D. Boneh, B. Bünz, and B. Fisch, ''Batching techniques for accumulators with applications to iops and stateless blockchains,'' in *Advances in Cryptology—CRYPTO*, A. Boldyreva and D. Micciancio, Eds. Cham, Switzerland: Springer, 2019, pp. 561–586. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-26948-7_20
[53] R. Gennaro, S. Halevi, and T. Rabin, ''Secure hash-and-sign signatures without the random oracle,'' in *Advances in Cryptology—EUROCRYPT '99*, vol. 1592. Cham, Switzerland: Springer, 1999, pp. 123–139.
[54] A. Fiat and A. Shamir, ''How to prove yourself: Practical solutions to identification and signature problems,'' in *Proc. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT)*. Berlin, Germany: Springer, 1986, pp. 186–194.
[55] (Oct. 2023). *Harchaoui Abdel Ali*. [Online]. Available: https://github.com/harchaoui/poeddp
[56] Accessed: Oct. 23, 2023. [Online]. Available: https://www.pycryptodome.org
[57] M. J. Dworkin, ''SHA-3 standard: Permutation-based hash and extendable-output functions,'' Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NIST FIPS 202, Jul. 2015.
[58] N. Bordes, J. Daemen, D. Kuijsters, and G. V. Assche, ''Thinking outside the superbox,'' in *Advances in Cryptology—CRYPTO*, 2021, pp. 337–367.
[59] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, ''Fast reed–solomon interactive oracle proofs of proximity,'' in *Proc. 45th Int. Colloq. Automata, Lang., Program. (ICALP)* (Leibniz International Proceedings in Informatics (LIPIcs)), vol. 107, I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, Eds. Dagstuhl, Germany: Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2018, pp. 14:1–14:17. [Online]. Available: https://drops-dev.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2018.14
[60] B. Yan et al., ''Factoring integers with sublinear resources on a superconducting quantum processor,'' 2022, *arXiv:2212.12372*.

ABDEL ALI HARCHAOUI is currently pursuing the Ph.D. degree with the Faculty of Science, Abdelmalek Essaâdi University, Morocco.

He worked as a Cryptography Researcher at (=nil;foundation) and was actively involved in the Placeholder proving system and Crypto3++ projects. He is interested in verifiable computation and formal verification of crypto libraries, using Dafny. His research interests include PLONKish arithmetization systems, comparing different optimizations used on the recent proving systems, such as Halo2, Plonky2, Kimchi, and Pickles from Mina protocol, Barretenberg from Aztec Protocol, and implementing custom PLONK circuits. He is also interested in HPC.

ALI YOUNES (Member, IEEE) received the Ph.D. degree in computer sciences from the Information and Telecommunication Systems Laboratory, Faculty of Science, Abdelmalek Essaâdi University, Morocco, in 2014.

He is currently a Professor of computer science with the Department of System Engineering, Faculty of Science, Morocco. He coauthored the book *Innovations in Smart Cities Applications Edition 2: The Proceedings of the Third International Conference on Smart City Applications* (Springer, 2019). His main research interests include distributed systems, heuristic algorithms, blockchain technology, grid and cloud computing, the IoT, and artificial intelligence. He has published several refereed research publications in these areas.

ABDELAAZIZ EL HIBAOUI received the master's degree in systems programming from Toulouse III-Paul Sabatier University, France, in 2002, and the Ph.D. degree from Bordeaux 1 University, France, in 2016.

From 2017 to 2022, he was the Head of the Computer Science and Systems Engineering Laboratory, Abdelmalek Essaâdi University, Morocco. He was also the Mathematics and Computer Science Stream Coordinator with Abdelmalek Essaâdi University, where he is currently a Full Professor with the Department of Computer Science, Faculty of Science. He supervised numerous undergraduate and graduate students in computer science. He has delivered courses across various computer science levels at universities in France and Morocco. He contributes to various peer-reviewed journals and mathematics and computer science conference proceedings. His research interests include randomized distributed algorithms, privacy and computer security, smart grids, artificial intelligence, machine learning, and informatics for renewable energy.

Prof. Hibaoui has undertaken roles, such as the Treasurer of the IEEE Morocco Section and the general chair, an organizing member, the session chair, and a Technical Program Committee member for numerous editions of esteemed national and international conferences.

AHMED BENDAHMANE (Member, IEEE) received the Ph.D. degree in computer sciences from Abdelmalek Essaâdi University, Morocco, in 2013.

From 2003 to 2016, he was the Head of IT Service with the Faculty of Letters, Abdelmalek Essaâdi University. Since 2016, he has been a Professor of computer sciences with Abdelmalek Essaâdi University. His main research interests include distributed systems, grid and cloud computing systems security, intrusion detection and tolerance, blockchain, and smart systems. He is the author and coauthor of several papers covering these and other topics, which appeared in refereed specialized journals and international conferences.