Big Data and Information Analytics

https://www.aimspress.com/journal/BDIA

BDIA, 9: 106–135.
DOI:10.3934/bdia.2025006
Received: 05 September 2025
Revised: 22 October 2025
Accepted: 28 October 2025
Published: 04 November 2025

Research article

Lightweight multi-copy public data auditing scheme with error localization for fog-assisted digital twins

Pan Yang*

School of Mathematical Sciences, Harbin Engineering University, Harbin 150001, China

* **Correspondence:** yangpan35@hrbeu.edu.cn.

Abstract: The integrity and durability of digital twin data determine whether virtual models can meet the requirements of continuous prediction and maintenance of physical entities throughout their lifecycle. Unfortunately, existing cloud data auditing schemes fail to balance achieving lightweight public verification and low communication overhead in multi-cloud and multi-copy scenarios. Additionally, the dual factors of the existence of multiple cloud service providers and multiple copies increase the difficulty of locating corrupted data when verification fails, while fine-grained error localization is rarely explored. To tackle these challenges, we propose a lightweight certificateless public data auditing scheme for multi-copy and multi-cloud scenarios in a fog-assisted digital twin system, which supports batch auditing and block-level error localization. Both computational cost and communication overhead during verification remain constant, and are independent of the number of copies and data blocks. Furthermore, the error localization mechanism enables the accurate and rapid identification of malicious cloud servers, damaged copies, and corrupted data blocks, and supports data recovery without downloading any copies. The security analysis confirms the soundness and unforgeability of the proposed scheme. Performance evaluations and comparative analyses indicate that our scheme is efficient and practical.

Keywords: public data auditing; lightweight; certificateless; multi-cloud; multi-copy; fog-assisted digital twins

1. Introduction

A digital twin (DT) is a digital representation of a physical object and is actively applied in the lifecycle management of products, equipment, and even plants. Currently, enormous DT data, such as modeling data, simulation outputs generated by virtual models, connectivity data linking virtual models and physical entities, and fusion data from different sources, is hosted on cloud servers (CSs) to obtain on-demand storage space and high-performance computing services [1, 2]. However, cloud data loss

could be caused by various reasons including operational errors by data owners, malicious deletion by cloud service providers (CSPs) due to contract disputes, unauthorized erasure of cold data by CSPs to resell storage space repeatedly, and external factors such as hacker attacks and force majeure. This is unbearable for data owners since most cloud-stored data lacks full local backups [3, 4]. Therefore, timely detection of cloud data corruption is critical to ensuring data integrity and availability.

Accessing the original file to check the integrity of the data incurs additional communication overheads and raises the risk of data leakage [5, 6]. To this end, there have been plenty of explorations into implementing blockless verification of remote data, such as those in [7–13]. Schemes [8–10] can remotely recover data when the number of corrupted blocks is below a specified threshold, but they are ineffective when the number of corrupted blocks exceeds the threshold. In order to avoid a single point of failure and ensure data availability, data owners prefer to upload multiple copies to different CSs, and protocols for multi-copy auditing have been extensively studied [14–24].

Schemes based on the public key infrastructure (PKI) [14, 16, 19] require complex certificate management, which is a challenging endeavor for DT systems with numerous devices. Although identity-based protocols [17, 18, 25] avoid this issue, they require a key generation center (KGC) to issue a private key for each data owner, which poses a key escrow problem. The key escrow issue not only necessitates KGC trustworthiness but also makes it more likely to entice malicious entities to conduct eavesdropping and attacks. Certificateless schemes [20–23] appear to be an promising solution.

Unfortunately, most existing multi-copy auditing schemes exhibit linearly increasing computational overhead in integrity verification, such as those in [17, 18, 21–23]. The high computational costs for data auditing require higher computing capacity from the third-party auditor (TPA) and may cause delays, as cloud data typically requires verification before use. Most studies on lightweight verification focus on single copies [26–28]. Although the scheme in [20] achieves lightweight auditing, the total communication overhead in the proof phase grows with the square of the number of copies. Delayed data responses and decision feedback affect the collaborative work between DT devices, and high communication overhead is barely tolerable for DT systems handling large-scale data.

In addition, how to accurately locate all corrupted blocks and remotely recover them remains an issue that needs to be addressed. In practice, batch verification of proofs from multiple CSs is often adopted to improve verification efficiency. However, an invalid proof can cause all verifications to fail. That is to say, a malicious CS can disrupt data integrity verification by providing invalid proofs. Therefore, locating errors is crucial for multi-cloud public auditing. Existing schemes [13, 26, 29] can only locate corrupted files. The ideal method is to find corrupted data blocks and recover them, rather than the entire file. Studies [20, 21] have proposed block-level error localization for multi-copy integrity verification in multi-cloud storage, but their methods cannot locate all corrupted data blocks comprehensively.

The physical entities in DT systems are equipped with sensors and controllers to collect data. On the one hand, this data reflects the status and performance of the edge devices; on the other hand, it is uploaded to cloud servers for in-depth analysis and optimization of the digital twin model. However, the communication overhead caused by frequent data transfers is a significant burden for resource-constrained devices. Fog computing has been widely adopted in many fields because it has higher computing and communication capabilities than resource-constrained devices and is closer to data sources than cloud servers. Introducing fog computing into DT systems helps reduce computational

burdens on edge devices and providing more efficient data preprocessing and transmission [30].

Therefore, how to design a lightweight and low-latency multi-copy integrity verification mechanism for fog-assisted DT systems and how to accurately locate all corrupted blocks remains a critical challenge yet to be tackled.

1.1. Contributions

In this paper, we propose a flexible and efficient multi-copy public auditing scheme for multi-cloud storage in DT systems, which supports fog-assisted data preprocessing, certificateless lightweight verification, batch auditing, and fine-grained error localization.

- We propose a lightweight certificateless public data auditing scheme for multi-copy and multi-cloud scenarios in a fog-assisted digital twin system (LPDA-MCMC). Compared to previous multi-copy public auditing schemes, our proposal achieves a significantly lower constant computational overhead for multi-copy verification while maintaining low communication overhead, both of which are independent of the number of copies and data blocks.
- In addition to lightweight verification, the proposed scheme is based on an end-fog-cloud three-layer architecture, which facilitates efficient and secure data preprocessing and uploading. In particular, the proposed scheme enables each fog node to preprocess data collected by terminal devices and generate copies along with homomorphic tags, which will subsequently be transferred to multiple cloud servers for long-term storage.
- Batch auditing and fine-grained error localization enhance the flexibility and practicality of the proposed scheme. Batch auditing allows a verifier to challenge all files and their copies generated within a specific time period. Fine-grained error localization can efficiently locate corrupted data blocks, enabling data recovery to concentrate on the corrupted blocks rather than the entire file, thus significantly reducing communication overhead.
- We formalize the security model. Security analysis demonstrates that the proposed scheme is unforgeable and achieves a high probability of detecting corrupted blocks. Meanwhile, it effectively resists copy-summation attacks, replay attacks, and replacement attacks. Performance evaluations and comparative analysis indicate that the proposed scheme is efficient and suitable for large-scale data verification.

1.2. Paper organization

The remainder of this paper is organized as follows: Section 2 reviews relevant works. Section 3 outlines the system model and formalizes the security model of LPDA-MCMC. Section 4 describes the concrete design of LPDA-MCMC. Security proofs and performance evaluations are respectively provided in Sections 5 and 6. Finally, Section 7 concludes this paper.

2. Related work

2.1. Certificateless public auditing

Extensive research progress has been made in certificateless public auditing for outsourced data. Wang et al. [31] presented the first certificateless data auditing scheme based on bilinear pairings in

2013, which not only maintains the advantage of certificate free management in [32] but also implements public blockless auditing. Following that, various certificateless public auditing protocols were explored [28, 33, 35–39]. He et al. [33] proposed a privacy-preserving certificateless provable data possession (PDP) in 2017. They calculated the tag to be $(x_O + s_O) \cdot (\bar{H}(id_i) + m_i \cdot \Psi_O)$ while the partial public key of the data owner O was $(x_O + s_O) \cdot \Psi_O$. It is obvious that their proposal did not achieve the unforgeability, as a portion of the tag can be calculated via public parameters. For specific security analysis, see Liao et al. [34]. Subsequently, Ji et al. [35] improved the algorithm and security model of the scheme in [33]. Recently, certificateless public integrity checking schemes suitable for group sharing of cloud data, and designated verifiers were studied [36–39]. The number of exponential operations for verification in most schemes grows linearly with the number of challenged blocks. To reduce computational overhead, Zhang et al. [28] presented a lightweight public auditing scheme through utilizing a homomorphic hash function to generate tags. The homomorphic properties and pre-computation make a constant computational cost for single-copy verification. Fast dynamic operations by improving the data structure in [40] and conditional anonymity are also implemented in [28].

2.2. Multi-copy auditing

In 2008, Curtmola et al. [14] established multi-replica data proof of possession (MR-PDP), the first attempt to enable a data owner to perform blockless verification on a copy hosted on a cloud server. Hao and Yu [15] subsequently proposed a publicly verifiable protocol, which improved the flexibility of MR-PDP. Liu et al. [16] constructed a multi-replica Merkle hash tree data structure for dynamic data in the cloud and developed a secure public data auditing protocol for multiple replicas. However, these schemes only allow the verifier to audit a single copy at a time. To achieve batch auditing for multiple copies, Barsum and Hasan [41] designed a provable multi-replica dynamic data possession scheme, see also [42]. Identity-based models for multi-copy and multi-cloud cases were proposed to simplify certificate management [17, 18, 25]. However, the use of the BLS signature [43] in [17] still makes managing certificates inevitable. Chang et al. [18] improved the scheme in [17] to propose a real identity-based model IB-MCMC, and introduced a map-to-point hash function to reduce the communication overhead during the data upload and verification phases. Due to the inherent key escrow issue in identity-based models, certificateless multi-copy public auditing schemes [22, 44] have been studied to overcome complex certificate management and key escrow issue simultaneously.

Unfortunately, the above schemes [17, 18, 22, 25, 41, 44] are vulnerable to the copy-summation attack [45], that is, a dishonest CSP can pass the verifier's challenge by only storing the sum of all copies of a data block. Guo et al. [45] provided a simple and effective solution by blinding each copy with a random number during the data auditing phase. Recently, Shen et al. [23] designed a certificateless PDP scheme termed CLPDP-MCMS and constructed a map version marker table to allow dynamic operations and historical data tracking for multiple copies of electronic medical records. Miao et al. [21] proposed a blockchain-assisted certificateless PDP scheme termed BAPDP-MCMC, which allows for block-level dynamic operations on copies by constructing a data structure and realizing fault localization. The schemes in [21, 23] are secure against the copy-summation attack, but the computational overhead during the verification phase increases linearly with the product of the number of corrupted blocks and the number of corrupted copies. Li et al. [20] used the hash function to realize lightweight verification for multiple copies and clouds cases,

but the communication overhead of the returned proof increases with the square of the number of copies.

2.3. Error localization

It is critical to identify the locations of incorrect data blocks when verification fails in multi-cloud scenarios, as the verifier typically checks the aggregation of all responses rather than individual ones. The straightforward approach is for the verifier to examine the received proofs one by one to identify malicious CSs, as proposed in [13, 26]. These two schemes focus on identifying invalid proofs through arbitration, which in turn enables the data owner to pinpoint malicious CSs in order to claim compensation. Su et al. proposed an error localization method for corrupted files, and achieved privacy-preserving data recovery by employing Cauchy matrix-based Reed-Solomon erasure-correcting codes (C-RSC) [29]. Li et al. implemented the localization of a random set of blocks. Due to the randomness of the set, this scheme cannot find all corrupted data blocks [20]. Zhao et al. employed matrix computations based on the uniform (K, N)-set and product-matrix minimum storage regeneration(PM-MSR) code to achieve file-level localization and recovery of erroneous data [46]. Due to the involvement of matrix inversion operations, its computational complexity is $O(m^3)$. The above schemes only offer a method to identify malicious CSs or corrupted files, without addressing the localization of corrupted data blocks. Moreover, their data recovery typically requires downloading the original file, which is inefficient for large files. By contrast, Miao et al. proposed a blockchain-assisted fault localization scheme that achieves fine-grained localization of corrupted copies and data blocks via smart contracts and binary search [21]. Specifically, the proof $P_{Copy_i}(P_i)$ related to $\lceil log_2 n \rceil$ corrupted copy indexes (or block indexes of a copy) derived from binary search, as well as $witness\_copy(witness\_block)$ consisting of the left side L_r and right side R_r of the corresponding verification equation of $P_{Copy_i}(P_i)$, serve as inputs to the smart contract, which then outputs the verification results. Notably, two implicit issues exist here: (1) There are corrupted copies or blocks left undetected when the number of corrupt copies or data blocks exceeds $\lceil log_2 n \rceil$. (2) If the smart contract verifies the correctness of L_r and R_r using $P_{Copy_i}(P_i)$ and public parameters, this results in redundant calculations for L_r and R_r. If not, malicious CSs could exploit the equality $L_r = e(\sigma_{Copy_i}, g) = R_r (L_r = e(\sigma_i, g) = R_r)$ to conceal the corruption of the copy (or data block). The comparative analysis of key attributes of relevant scheme is detailed in Table 1.

Table 1. Comparative analysis of key attributes.

Schemes	Basis	Error Localization	Methods	Data Recovery
(2022) [13]	RSA	File-level	Arbitration	–
(2022) [26]	RSA	File-level	Arbitration	–
(2022) [21]	Bilinear pairing	Block-level	Binary search	$O(k)$
(2022) [29]	Bilinear pairing	File-level	$(v, k+1, 1)$-design and (v, m) C-RSC	$O(m^2 n)$
(2023) [20]	Hash function	File-level	Arbitration	–
(2024) [46]	Bilinear pairing	File-level	Uniform (K, N)-Set and PM-MSR code	$O(m^3)$
Ours	Bilinear pairing	Block-level	Recursive binary search	$O(k)$

Note: *Methods* are the approaches for error localization; k denotes the number of corrupted blocks; m and n denote the number of original data components and the number of data blocks per component, respectively.

3. Preliminaries and system model

3.1. Notations and preliminaries

Table 2 decribes the notations in this paper.

Table 2. Notations.

Notations	Descriptions
G_1, G_2	Multiplicative cyclic group with order p
p	A large prime
H_1, H_2	Hash function
π	Pesudorandom permutation
f_1, f_2, f_3	Pesudorandom function
$\mathrm{CS_k}$	The identifier of a cloud server
$m_{l,i,j}$	The l-th copy of j-th sector of i-th block
ξ	The cloud server number
r	The copy number
n	The block number of file F
s	The sector number of a block
c	The number of challenged blocks
$\|G_1\|$	The length of an element in G_1
$\|p\|$	The length of an element in Z_p^*
$\|H\|$	Map to point hash operation
$\|h\|$	Hash operation
E_{G_1}	Exponential operation in G_1
$Pair$	The bilinear pairing operation

Definition 1. *(Bilinear pairing [43]) Let G_1, G_2 be two different multiplicative cyclic groups of prime order p, $e : G_1 \times G_1 \to G_2$ is called a bilinear pairing map if it satisfies the following properties:*

(1) Bilinearity: $e(g^a, v^b) = e(g, v)^{ab}$, $\forall\ a, b \in Z_p^*$, $g, v \in G_1$;

(2) Non-degeneracy: $\exists\ g, v \in G_1$ such that $e(g, v) \neq 1$;

(3) Computability: $\forall\ a, b \in Z_p^*$, $\forall\ g, v \in G_1$, $e(g^a, v^b) = e(g, v)^{ab}$ can be computed in probability polynomial time (PPT).

Definition 2. *The discrete logarithm (DL) assumption [21] implies that the probability of any PPT adversary solving α from the tuple (g, g^α) is negligible, where g is a generator of the multiplicative group G_1 of prime order p and $\alpha \in Z_p^*$ is an unknown random number.*

Definition 3. *The computational Diffie-Hellman (CDH) assumption [23] implies that the probability of any PPT adversary solving $g^{\alpha b}$ from the tuple (g, g^α, g^b) is negligible, where g is a generator of the multiplicative group G_1 of prime order p and $\alpha, b \in Z_p^*$ are unknown random numbers.*

3.2. *System model*

We assume that a fog-assisted DT system is divided into multiple coverage areas, each containing a number of physical entities equipped with sensors and controllers. Each coverage area is deployed with a fog computing node to generate copies and homomorphic verifiable tags for the collected data, which are transmitted to multiple CSs for long-term storage or provided to applications. The system model of LPDA-MCMC is illustrated in Figure 1.

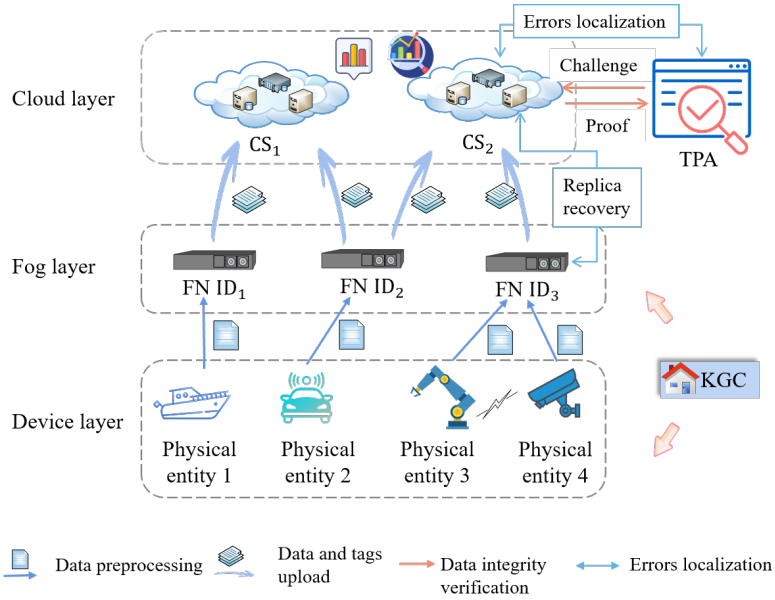


Figure 1. System model.

Physical entity (PE): PE is equipped with sensors and controllers to provide real-time data for its digital twin. The data is uploaded to the CSs for long-term storage after being preprocessed by FN. All PEs constitute the device layer to execute tasks and collect real-time data.

Fog computing node (FN): FN serves as the core hub of the end-fog-cloud architecture. FN offloads computational tasks for PEs and performs data preprocessing; generates homomorphic tags to ensure data auditability; coordinates interactions between PEs, CSs, and the TPA; optimizes transmission latency and auditing overhead; and assists in corrupted block recovery. It adapts to the low-power consumption and real-time requirements of DT systems, thua acting as a key enabler for lightweight auditing. All FNs constitute the fog layer.

CSs: CSs refer to cloud servers that belong to multiple cloud service providers and have massive storage and powerful computing resources. CSs are responsible for providing proof of data possession to demonstrate that the challenged data is complete. All CSs constitute the cloud layer.

KGC: The KGC holds the system master secret key and is responsible for generating system public parameters as well as partial private keys for members of the system.

TPA: The TPA refers to a third-party auditor who is responsible for conducting periodic data auditing and may be curious regarding data content.

In this paper, we assume that the CSs are semi-honest parties who provide storage service but may conceal data corruption to whitewash their reputations and avoid compensation, and the TPA will honestly perform verification tasks but is highly curious about the audited data.

The proposed scheme consists of the following algorithms.

Setup: The KGC runs this algorithm to generate the master secret key x and system public parameters $para$, taking the secure parameter 1^λ as input.

ParPriKeyGen: The KGC runs this algorithm to generate the partial private key θ for each FN in the systems by using x, $para$, and the FN's identity ID as input.

KeyGen: Each FN runs this algorithm to generate the private key sk_{ID} and public key with $para, \mathrm{ID}, \theta$ as input.

Preprocessing: First, the PE runs this algorithm to encrypt data blocks. Then the FN generates r copies of the encrypted data blocks immediately upon receiving the encrypted blocks.

TagGen: Each FN runs this algorithm to generate tags for all data blocks by using $para, sk_{\mathrm{ID}}, F$ as input.

ChalGen: The TPA runs this algorithm to output a challenge by taking $para$ as input.

ProofGen: The CS_k runs this algorithm to generate a proof P_k in response to the challenge with ID, $\{F_l\}_{l=1,\cdots,r}$, $para$, $Chal$ as input.

ProofVer: The TPA takes ID, P_k, $para$ as input and verifies the correctness of the stored challenged data blocks by running this algorithm.

3.3. Security model

The basic security of LPDA-MCMC includes two points: (a) correct data can always pass verification; and (b) unforgeability holds under the DL and CDH assumptions. Let $\mathcal{A}, C$, $\widetilde{\mathrm{ID}} \in \{0,1\}^*, \tilde{m} = (m_1, m_2, \cdots, m_s)$ be the adversary, the challenger, the challenged identity, and challenged data block, respectively. We first consider a series of actions that a general adversary may take to against the proposed scheme.

(1) **Partial Private Key Extraction:** $\mathcal{A}$ is allowed to issue queries for the partial private key of an entity of its choice. Let the entity's identity be ID. C generates the partial private key for ID in response by running the **ParPriKeyGen** algorithm.

(2) **Private Key Extraction:** $\mathcal{A}$ can extract the private key for an entity ID of its choice, except when the entity's public key has been replaced. C responds to this request by running the **KeyGen** algorithm (by first running **ParPriKeyGen** if necessary).

(3) **Request for Public Key:** The public key for any ID is available to $\mathcal{A}$. C runs algorithm **PubKeyGen** to generate the requested public key (by first running **KeyGen** if necessary).

(4) **Replace Public Key:** Since there is no certificates in the proposed scheme, it is possible to replace the public key. $\mathcal{A}$ is allowed to replace the public key pk for an ID with a new value pk'. This replacement can be repeated. The value pk' will then be used as the public key in subsequent calculations.

(5) **Tag queries:** $\mathcal{A}$ queries the tag of the data block m associated with an identity ID and its public key pk. If $m \neq \tilde{m}, \mathrm{ID} \neq \tilde{\mathrm{ID}}$, and $pk \neq pk_{\tilde{\mathrm{ID}}}$, C returns the corresponding tag as a response.

Due to the adoption of certificateless public keys in our proposed scheme, two types of adversaries need to be considered. The Type I adversary, denoted by $\mathcal{A}_1$, is allowed to replace the public key of its choice but does not know the master secret key. Such adversary exists and may be a malicious CS. Two

natural restrictions apply to $\mathcal{A}_1$: (a) for the challenged identity $\widetilde{\mathrm{ID}}$, $\mathcal{A}_1$ is not allowed to replace the public key and extract its partial private key or private key; and (b) if the public key of an entity ID has been replaced, $\mathcal{A}_1$ cannot query the corresponding private key for ID. The Type II adversary denoted by $\mathcal{A}_2$, knows the system master secret key but cannot replace any entity's public key, for example, an eavesdropping KGC. $\mathcal{A}_2$ cannot extract the private key of $\widetilde{\mathrm{ID}}$.

Now we provide a formal definition of security to ensure the security of LPDA-MCMC.

Definition 4. *The proposed scheme is secure and unforgeable if no polynomial-bounded adversary ($\mathcal{A}_1$ or $\mathcal{A}_2$) has a non-negligible advantage against the challenger (C) in the following games.*

Phase 1. *C initializes the system and publishes public parameters. Note that the master secret key is unknown to $\mathcal{A}_1$ but known to $\mathcal{A}_2$.*

Phase 2. *In this phase, the adversary can make a series of queries. Specifically, $\mathcal{A}_1$ is allowed to perform the above five predefined types of queries, while $\mathcal{A}_2$ is allowed to perform **Private Key Extraction**, **Request for Public Key**, and **Tag queries**.*

Phase 3. *This phase consists of two cases:*

(1) The adversary $\mathcal{A}_1$ is required to forge a proof for a set of challenged blocks with $\widetilde{\mathrm{ID}}$.

(2) The adversary $\mathcal{A}_2$ is required to forge a tag for the challenged block $\tilde{m}$ with $\widetilde{\mathrm{ID}}$.

The adversary $\mathcal{A}_1$ (or $\mathcal{A}_2$) wins the game if the forged proof (or tag) is valid in Phase 3.

Note that $\mathcal{A}_2$ can extract the partial private key of all identity of its choice by utilizing the master secret key.

3.4. Design goals

- Lightweight certificateless public auditing: Realizing certificateless multi-copy public auditing in multi-cloud storage, and the computational cost in the verification phase reduces to a constant value.
- Batch auditing: Allowing the TPA to batch audit all copies of multiple files distributed across multiple CSs in a single data auditing task.
- Error localization: Supporting fine-grained localization to identify corrupted data blocks.
- Fog-assisted: Introducing fog computing nodes to support low-latency data preprocessing and uploading.
- Enhanced security: In addition to the aforementioned security properties, the proposed scheme is secure against the copy-summation attack, the replay attack, and the replacement attack.

4. Concrete construction

4.1. The proposed scheme

In this subsection, we present the detailed construction of LPDA-MCMC, including **Setup**, **ParPriKeyGen**, **KeyGen**, **Preprocessing**, **TagGen**, **Chal**, **ProofGen**, and **ProofVer**. The work flow of LPDA-MCMC is shown in Figure 2.

Setup: Given the secure parameter 1^{λ}, the KGC selects a finite field Z_p, and two distinct multiplicative cyclic groups G_1 and G_2 of order p. Let $e : G_1 \times G_1 \to G_2$ be a bilinear pairing, and g, u

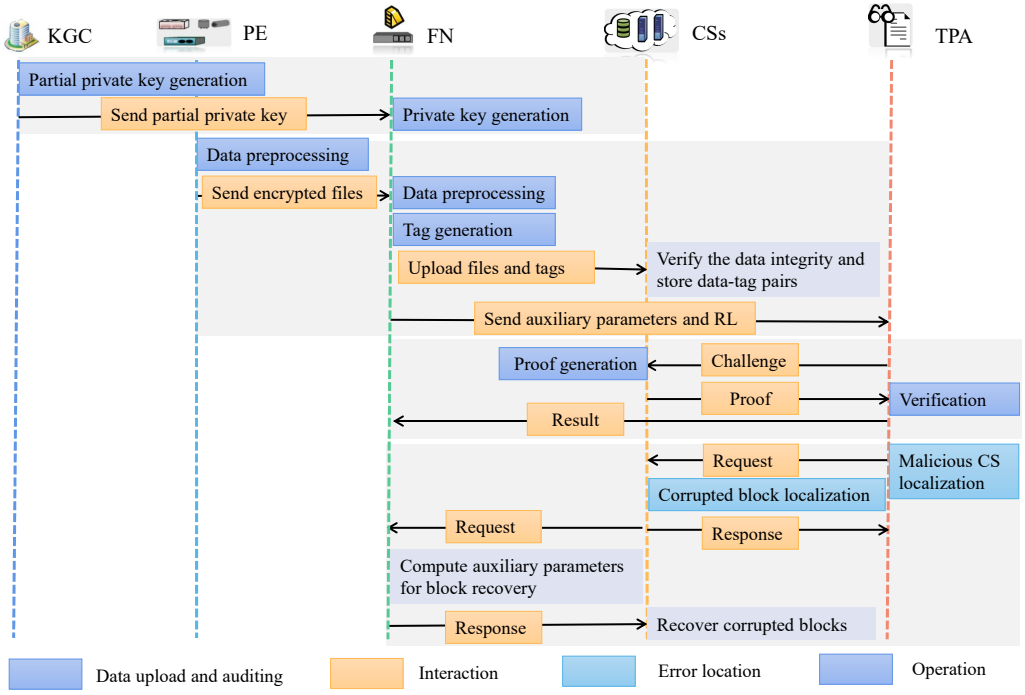


Figure 2. Work flow.

be two different generators for group G_1. The KGC further selects two hash functions $H_1 : \{0,1\}^* \to G_1$ and $H_2 : \{0,1\}^* \to Z_p^*$, one pseudo-random permutation $\pi : Z_p^* \times \{1,2,\cdots,n\} \to \{1,2,\cdots,n\}$, and three pseudo-random functions $f_i : Z_p^* \times \{0,1\}^* \to Z_p^*$, $i = 1,2,3$, where f_1 maps to the copy space, f_2 maps to the random number set for challenged block indexes, and f_3 maps to the random number set for challenged copy indexes. The KGC randomly selects $\alpha \in Z_p^*$ as the master secret key, computes $x = g^\alpha$, and publishes the public parameters $para = \{e, G_1, G_2, Z_p^*, H_1, H_2, \pi, f_1, f_2, f_3, g, u, x\}$.

ParPriKeyGen: Upon receiving the request from the FN with identity ID, the KGC computes $\theta = H_1(\mathrm{ID})^\alpha$ as FN's partial private key by utilizing the master secret key and FN's identity ID. The KGC sends θ to the FN in a secure channel.

KeyGen: Once receiving θ, each FN first checks the partial private key θ as follows

$$e(g,\theta) \stackrel{?}{=} e(x, H_1(\mathrm{ID})). \tag{4.1}$$

If the equality holds, select $\gamma \in Z_p^*$ randomly. Let $sk_{\mathrm{ID}} = (\gamma, \theta)$ be the private key, and compute $pk_{\mathrm{ID}} = (pk_1, pk_2) = (x^\gamma, g^\gamma)$ as the public key. Note that

$$e(g,\theta) = e(g, H_1(\mathrm{ID})^\alpha) = e(x, H_1(\mathrm{ID})).$$

Preprocessing: This algorithm is executed sequentially by the PE and the FN. Let F_{name} be the identifier of collected file F.

a) The PE divides the file into n data blocks and encrypts each data block through utilizing symmetric encryption. Subsequently, the PE sends the encrypted blocks $m_1, \cdots, m_n$ to FN for further processing, where $m_i \in Z_p^*, i = 1,2,\cdots,n$.

b) The FN selects a random $\psi \in Z_p^*$ and generates r copies $\{F_l\}_{l=1,\cdots,r}$ for F. For l-th copy $m_{l,i}$ of block m_i, compute $m_{l,i} = m_i + f_1(\psi, F_{name}||l||i) \pmod p$, and fragment the block into s sectors. Set $m_{l,i} = (m_{l,i,1}, \cdots, m_{l,i,s})$, $F_l = \{m_{l,1}, \cdots, m_{l,n}\}$.

TagGen: The FN randomly selects $\lambda_j \in Z_p^*$ and computes $u_j = u^{\lambda_j},\ j = 1, 2, \cdots, s$. For each data block $m_{l,i}$, compute the tag as

$$T_{l,i} = (\theta^{h_{l,i}} \cdot u^{\sum_{j=1}^{s} \lambda_j m_{l,i,j}})^{\gamma}, l = 1, \cdots, r,\ i = 1, \cdots, n,$$

where $h_{l,i} = H_2(\mathrm{CS}_k||F_{name}||l||i||t)$, CS_k is the identity of designated cloud server. Let

$$\omega_l = \mathrm{ID}||\mathrm{CS}_k||F_{name}||l||n||t||u_1||\cdots||u_s||pk_{ID}||Sig(\mathrm{ID}||\mathrm{CS}_k||F_{name}||l||n||t||u_1||\cdots||u_s||pk_{ID}),$$

where $k \in \{1, \cdots, \xi\}$, Sig is a secure certificateless signature. Then the FN uploads $\Phi_l = \{(m_{l,i}, T_{l,i})\}_{i=1,\cdots,n}, \omega_l$ to CS_k, which will store the data-tag pairs only if Eq (4.2) holds.

$$e(g, T_{l,i}) \stackrel{?}{=} e(pk_1, H(\mathrm{ID})^{h_{l,i}}) \cdot e(pk_2, \prod_{j=1}^{s} u_j^{m_{l,i,j}}). \tag{4.2}$$

Then FN sends auxiliary parameters ω_l to the TPA, which adds the tuple $(\mathrm{ID}, t, F_{name}, l, \mathrm{CS}_k),\ l = 1, \cdots, r,\ k = 1, \cdots, \xi$ into the record list (RL).

Chal: When the TPA is required to verify the integrity of F, it first searches for the locations $\{\mathrm{CS}_k\}_{k=1,\cdots,\xi}$ of all copies of F from RL. Then randomly select $k_1, k_2, k_3 \in Z_p^*$, an integer $c \in [1, n]$, grab the current time stamp $\hat{t}$, and distribute the challenge $(k_1, k_2, k_3, c, \hat{t})$ to the corresponding $\{\mathrm{CS}_k\}_{k=1,\cdots,\xi}$.

Next the TPA calculates auxiliary parameters for subsequent verification. Specifically, compute the index set $I = \{\pi(k_1, \hat{t}||c||i)|i = 1, 2, \cdots, c\}$, $v_i = f_2(k_2, \hat{t}||c||i) \in Z_p^*$ for $i \in I$, and $f_3(k_3, \hat{t}||l) = a_l$ for $l = 1, \cdots, r$. Parse the parameters $\mathrm{ID}, \mathrm{CS}_k, F_{name}, t, u_1, \cdots, u_s, pk_{\mathrm{ID}}$ and get $\delta = \sum_{l=1}^{r} \sum_{i \in I} a_l H_2(\mathrm{CS}_k||F_{name}||l||i||t)v_i$.

ProofGen: Once receiving the challenge, the cloud server CS_k generates the partial proof P_k as follows.

(a) First compute the index set $I = \{\pi_{k_1}(\hat{t}||c||i)|i = 1, \cdots, c\}$. For each $i \in I$, compute $v_i = f_2(k_2, \hat{t}||c||i) \in Z_p^*$. For each copy F_l stored on CS_k, compute the random parameter $f_3(k_3, \hat{t}||l) = a_l$. Let the R_k represents the copy index set of the file F stored on CS_k. Note that all R_k are non empty sets and satisfy

$$\cup_{k=1}^{\xi} R_k = \{1, 2, \cdots, r\}, \quad R_i \cap R_j = \emptyset,\ \forall i \neq j,\ i, j \in \{1, \cdots, \xi\}.$$

(b) For each $l \in R_k$, compute

$$\sigma_l = (\prod_{i \in I} T_{l,i}^{v_i})^{a_l},\ \ \mu_{j,l} = a_l \sum_{i \in I} m_{l,i,j} v_i,\ \ j = 1, \cdots, s,$$

(c) Aggregate all $\sigma_l, \mu_{j,l}$ as

$$\sigma_k = \prod_{l \in R_k} \sigma_l,\ \ \mu_{j,k} = \sum_{l \in R_k} \mu_{j,l},\ \ j = 1, \cdots, s.$$

(d) Return the proof $P_k = (\sigma_k, \{\mu_{1,k}, \cdots, \mu_{1,s}\})$ to the TPA as the response.

ProofVer: Once receiving all responses $\{P_1, \cdots, P_\xi\}$, the TPA checks whether the challenged data blocks are stored correctly through the following computations.

(a) First compute

$$\sigma = \prod_{k=1}^{\xi} \sigma_k, \quad \mu_j = \sum_{k=1}^{\xi} \mu_{j,k}, \quad j = 1, 2, \cdots, s.$$

(b) If the equality (4.3) holds, the data is correctly stored in the CS_k, and the TPA returns "1"; otherwise, there is at least one corrupted data block, and the TPA returns "0".

$$e(g, \sigma) = e(pk_1, H_1(\mathrm{ID})^\delta) \cdot e(pk_2, \prod_{j=1}^{s} u_j^{\mu_j}). \tag{4.3}$$

Remark 1. *Without loss of generality, we consider c distinct blocks during the aforementioned verification process mentioned above. Suppose there are β corrupted data blocks in one copy. The probability of detecting at least one corrupted data blocks is* $1 - \prod_{i=0}^{c-1} \frac{n-\beta-i}{n-i} \geq 1 - (\frac{n-\beta}{n})^c$. *Therefore, the value of c should be at least* $\log_{1-\beta/n}(1 - Pr)$ *to achieve a detection probability of Pr for corrupted blocks.*

4.2. Batch auditing

We extend the proposed scheme to support more flexible batch verification, allowing the TPA to verify the integrity of the files and their r copies uploaded by ID within a time period T. Let the file index for these files be denoted as D.

Let the index set of the challenged files stored on CS_k be denoted as D_k. The TPA sends the challenges $(k_1, k_2, k_3, c_d, \hat{t})$ for $d \in D_k$, along with the copy index set $R_{d,k}$ of the challenged files, to the $\mathrm{CS}_1, \cdots, \mathrm{CS}_\xi$ according to the RL. All sets D_k and $R_{d,k}, k = 1, \cdots, \xi$ are non-empty and satisfy

$$\begin{aligned} &D = \cup_{k=1}^{\xi} D_k, \quad D_k \cap D_s = \emptyset, \ \ \forall k \neq s, \\ &\cup_{k=1}^{\xi} R_{d,k} = \{1, 2, \cdots, r\}, \ \ \forall d \in D_k, \\ &R_{d,i} \cap R_{d,j} = \emptyset, \forall i \neq j, \ i, j \in \{1, 2, \cdots, \xi\}. \end{aligned}$$

Compute the index set $I_d = \{\pi(k_1, \hat{t}||c_d||i)|i = 1, 2, \cdots, c_d\}$, $v_{d,i} = f_2(k_2, \hat{t}||c_d||i) \in Z_p^*$ for $i \in I_d$, and $f_3(k_3, \hat{t}||l) = a_l$ for $l \in \{1, \cdots, r\}$. The TPA calculates

$$\delta^{(T)} = \sum_{d \in D} \sum_{l=1}^{r} \sum_{i \in I_d} a_l H_2(\mathrm{CS}_k||F_{name}||l||i||t) v_{d,i}.$$

The cloud server CS_k is required to generate a partial proof $P_k = (\sigma_k, \{\mu_{1,k}, \cdots, \mu_{s,k}\})$ immediately after receiving the challenge. For each file index $d \in D_k$, it computes

$$\sigma_k = \prod_{d \in D_k} \prod_{l \in R_{d,k}} (\prod_{i \in I_d} T_{d,l,i}^{v_{d,i}})^{a_l},$$

$$\mu_{j,k} = \sum_{d \in D_k} \sum_{l \in R_{d,k}} a_l \sum_{i \in I_d} m_{d,l,i,j} v_{d,i}, \quad j = 1, 2, \cdots, s.$$

CS_k outputs the proof $P_k = (\sigma_k, \{\mu_{1,k}, \cdots, \mu_{s,k}\})$. The TPA checks if the challenged files are stored correctly, as shown in Algorithm 1. Equation (4.4) is given as follows.

Algorithm 1: Batch auditing

Input: $((k_1, k_2, k_3, c_d, \hat{t}), para, \{P_1, \cdots, P_\xi\})$
Output: $\{0, 1\}$

```
Initialize the σ^(T) as the identity element of G_1, μ_j^(T) as the zero element of finite field Z_p
for 1 ≤ k ≤ ξ do
    σ^(T) ← σ^(T) · σ_k
    for 1 ≤ j ≤ s do
        μ_j^(T) ← μ_j^(T) + μ_{j,k}
    end
end
Verify the proof (σ^(T), μ_j^(T)) as follows:
if Equation (4.4) holds then
    return 1
else
    return 0
end
```

$$e(g, \sigma^{(T)}) = e(pk_1, H_1(ID)^{\delta^{(T)}}) \cdot e(pk_2, \prod_{j=1}^{s} u_j^{\mu_j^{(T)}}) \tag{4.4}$$

4.3. Error localization

When **ProofVer** outputs 0, it indicates that there may be corrupted data blocks, and a set of steps must be taken to promptly identify and restore the corrupted data blocks to ensure data availability. Given that the number of copies and related CSs in practice is usually small, the computational resources consumed by traversal search to locate malicious CSs and corrupted copies are acceptable. However, numerous data blocks are challenged in a single auditing task to confirm that the data is corrected stored. Using traversal search to locate damaged blocks will result in a computational complexity of $O(rc)$. Thus, we develop a fine-grained error localization algorithm based on binary search and recursion. The details are as follows:

Step 1. Locate malicious CSs. The TPA verifies all responses $P_1, \cdots, P_\xi$ received in the current auditing task sequentially. If the proof P_k is invalid, the malicious CS_k is identified. The TPA then requires CS_k to provide a set of corrupted block indices.

Step 2. Locate the corrupted data block. The CS_k first distinguishes the incorrect copy l by checking $(\mu_{l,1}, \cdots, \mu_{l,s}, \sigma_l)$, $l \in R_k$ using Eq (4.3) one by one. Then for all incorrect copy F_l, $l \in R'_k$, the CS_k uses Algorithm 2 to output the index set E_k of all corrupted blocks, where R'_k consists of the index of incorrect copies. Subsequently, the CS_k sends E_k and auxiliary parameters $\sigma'_k, \mu_{1,k}, \cdots, \mu_{s,k}$ to the TPA.

Algorithm 2: Corrupted block location

Input: The index set I of for corrupted file F
Output: Error block index set E

```
Define errorLocation(I)
len = I.length
if len == 1 then
    return I
else
    Divide I into left subset L with length ⌈len/2⌉ and right subset R with length len − ⌈len/2⌉
    Initialize E as an empty set
    if Ver(L) = False then
        ls = errorLocation(L)
        E += ls
    else
        L is correct
    end

    if Ver(R) = False then
        rs = errorLocation(R)
        E += rs
    else
        R is correct
    end

    return E
end
```

$$\sigma'_k = \prod_{(l,i)\in R_k\times I-E_k} T_{l,i}^{a_l v_i}, \quad \mu'_{j,k} = \sum_{(l,i)\in R_k\times I-E_k} m_{l,i,j} a_l v_i, \quad j = 1, \cdots, s.$$

where $R_k \times I - E_k$ denotes all elements in the Cartesian product $R_k \times I$ except those that belong to $(R_k \times I) \cap E_k$.

Step 3. The TPA first verifies the response from CS_k,

$$\begin{aligned} e(g, \sigma'_k) =& e(pk_1, H_1(\mathrm{ID})^{\sum_{(l,i)\in R_k\times I-E_k} h_{l,i} a_l v_i}) \cdot \\ & e(pk_2, \prod_{j=1}^{s} u_j^{\sum_{(l,i)\in R_k\times I-E_k} \mu'_{j,k}}), \end{aligned}$$

and if it passes, it indicates that the corrupted block index set is valid. Note that

$$\begin{aligned} e(g, \sigma'_k) =& \prod_{(l,i)\in R_k\times I-E_k} e(g, T_{l,i}^{a_l v_i}) \\ =& e(pk_1, H_1(\mathrm{ID})^{\sum_{(l,i)\in R_k\times I-E_k} h_{l,i} a_l v_i}) \cdot \\ & e(pk_2, \prod_{j=1}^{s} u_j^{\sum_{(l,i)\in R_k\times I-E_k} \mu'_{j,k}}). \end{aligned}$$

Data recovery: The data is recovered as follows.

Case 1: If there exists a correct copy (l_o, i) stored on CS_k for $(l, i) \in E_k$, the FN computes

$$f_{ij} = f_1(\psi, F_{name}||l||i||j) - f_1(\psi, F_{name}||l_o||i||j) \pmod{p}.$$

Then CS_k immediately computes $m_{l,i,j} = f_{i,j} + m_{l_o,i,j} \pmod{p}$ for $(l, i) \in E_k$.

Case 2: If there is no available copy for $(l, i) \in E_k$ on CS_k, the FN downloads a correct copy $m_{l_o,i} = (m_{l_o,i,1}, \cdots, m_{l_o,i,s})$, and calculates

$$m_{l,i,j} = f_1(\psi, F_{name}||l||i||j) - f_1(\psi, F_{name}||l_o||i||j) + m_{l_o,i,j} \pmod{p}.$$

The FN may choose another CS to replace CS_k. In this case, the entire copy and its tag set need to be uploaded to this new CS. The data blocks will be stored after verified via Eq (4.2).

4.4. Correctness

It is proven here that the Eqs (4.2)–(4.4) in the proposed algorithm are valid, which indicates the correctness of the proposed scheme.

For Eq (4.2),

$$\begin{aligned} e(g, T_{l,i}) &= e(g, (\theta^{h_{l,i}} \cdot u^{\sum_{j=1}^{s} \lambda_j m_{l,i,j}})^{\gamma}) \\ &= e(g^{\gamma}, H_1(\mathrm{ID})^{\alpha h_{l,i}}) \cdot e(g^{\gamma}, \prod_{j=1}^{s} u_j^{m_{l,i,j}}) \end{aligned}$$

$$= e(pk_1, H_1(ID)^{h_{l,i}}) \cdot e(pk_2, \prod_{j=1}^{s} u_j^{m_{l,i,j}}).$$

For Eq (4.3),

$$\begin{aligned} e(g, \sigma) =& e(g, \prod_{k=1}^{\xi} \prod_{l \in R_k} (\prod_{i \in I} (\theta^{h_{l,i}} u^{\sum_{j=1}^{s} \lambda_j m_{l,i,j}})^{\gamma v_i})^{a_l}) \\ =& e(g^{\gamma}, \prod_{l=1}^{r} \prod_{i \in I} (H_1(\mathrm{ID})^{\alpha a_l h_{l,i} v_i} \cdot \prod_{j=1}^{s} u_j^{a_l m_{l,i,j} v_i})) \\ =& e(pk_1, H_1(\mathrm{ID})^{\delta}) \cdot e(pk_2, \prod_{j=1}^{s} u_j^{\mu_j}) \end{aligned}$$

For Eq (4.4),

$$\begin{aligned} e(g, \sigma^{(T)}) =& e(g, \prod_{k=1}^{\xi} \prod_{d \in D_k} \prod_{l \in R_{d,k}} (\prod_{i \in I_d} T_{d,l,i}^{v_{d,i}})^{a_l}) \\ =& e(g^{\gamma}, \prod_{l=1}^{r} \prod_{d \in D} \prod_{i \in I_d} T_{d,l,i}^{v_{d,i} a_l}) \\ =& e(pk_1, H_1(ID)^{\delta^{(T)}}) \cdot e(pk_2, \prod_{j=1}^{s} u_j^{\mu_j^{(T)}}). \end{aligned}$$

5. Security analysis

In this section, we discuss the security of the proposed scheme from four aspects: unforgeability, copy-summation attack resistance, replay attack resistance, and replacement attack resistance. Detailed proofs are provided in the Appendix.

Lemma 1. *Let H_1, H_2 be random oracles. If the CDH assumption holds, it is computationally infeasible for a Type I adversary to construct a forged tag for a given challenged block.*

Lemma 2. *Given a challenge $\{(\tilde{m}_1, v_1), \cdots, (\tilde{m}_i, v_i)\}$ for an identity $\widetilde{\mathrm{ID}}$, the related tags have never been queried prior to the challenge phase. If the CDH and DL assumptions hold, it is computationally infeasible for a Type I adversary to forge a valid proof that passes the TPA's verification.*

Lemma 3. *Let H_1, H_2 be random oracls. If the CDH assumption holds, it is computationally infeasible for a Type II adversary to construct a forged tag for a given challenged block.*

It should be noted that a Type II adversary could be a malicious KGC whose goal is to forge valid tags but not valid proofs. From Lemmas 1–3, we summarize Theorem 1 as follows.

Theorem 1. *Let H_1, H_2 be random oracles. If both CDH and DL assumptions hold, the proposed scheme achieves unforgeability.*

Theorem 2. *The proposed scheme is secure against copy-summation attacks.*

Theorem 3. *The proposed scheme is resistant to replay attacks. This ensures that any malicious CS efforts to replay the valid proof generated in previous interactions to respond to the current challenge will be detected.*

Theorem 4. *The proposed scheme is secure against replacement attacks. That is, any malicious CS cannot pass verification by replacing corrupted data blocks (or their corresponding proof) with legitimate data blocks (or valid proof).*

Table 3. Functional comparison.

Schemes	Type	Computation	F1	F2	F3	F4	F5	F6
[13]	PKI	$O(c\xi)$	✓	×	×	✓	×	–
[26]	PKI	$O(1)$	×	×	✓	✓	×	–
[18]	ID-based	$O(rc)$	✓	✓	×	×	×	×
[21]	Certificateless	$O(rc)$	✓	✓	×	✓	×	✓
[23]	Certificateless	$O(rc)$	✓	✓	×	×	×	✓
[46]	ID-based	$O(1)$	✓	×	✓	✓	×	–
Ours	Certificateless	$O(1)$	✓	✓	✓	✓	✓	✓

Note: *Computation* denotes the computational cost for verification; F1 denotes public auditing; F2 denotes multi-copy; F3 denotes batch auditing for several different files; F4 denotes error localization; F5 denotes fog-assisted; F6 denotes that the scheme can resist the copy-summation attack.

Table 4. Communication overhead (bits).

Schemes	KGC to DO/FN	TPA to CS	CS to TPA
IB-MCMC [18]	$\|G_1\|$	$(\xi+1)(log_2 n + 2\|p\|)$	$(\xi+2)\|G_1\| + (\xi+1)s\|p\|$
BAPDP-MCMC [21]	$\|G_1\|$	$(\xi+1)(log_2 n + 3\|p\|)$	$(\xi+1)(\|G_1\| + s\|p\|)$
CLPDP-MCMS [23]	$\|G_1\|$	$\xi(log_2 n + 3\|p\|)$	$\xi(\|G_1\| + s\|p\|)$
Our scheme	$\|G_1\|$	$\xi(log_2 n + 3\|p\|)$	$\xi(\|G_1\| + s\|p\|)$

6. Performance analysis

In this section, we compare the functional aspects of our proposed scheme to related data auditing protocols for multi-cloud storage, and then evaluate the performance of the proposed scheme from both theoretical and experimental perspectives. The definition of all notations is shown in Table 1.

6.1. Functional comparison

The proposed scheme is compared with the relevant multi-cloud storage data auditing schemes [13, 18, 21, 23, 26] as shown in Table 3. Obviously, our scheme is more efficient and practical for DT systems involving massive amounts of devices and data.

6.2. Theoretical analysis

This subsection discusses the storage, communication, and computational overheads of our proposed scheme.

Storage overhead. The storage overhead of our proposed scheme is mainly derived from storing data-tag pairs, with the overhead given by $rn(|G_1| + s|p|)$. Suppose that the element sizes in G_1 and Z_p^* are 512 bits and 160 bits, respectively. Setting $s = 50$, with a block size of 1 KB, a 10 MB file only 640 KB of storage for tags, not exceeding 6.4% of the required storage space for saving all data-tag pairs. In addition, the auxiliary parameters occupy no more than 4 KB of storage.

We compare our proposed scheme with closely related multi-copy multi-cloud schemes IB-MCMC [18], BAPDP-MCMC [21], and CLPDP-MCMS [23] in terms of communication and computational overheads.

Communication overhead. The communication between the KGC and the DO/FN occurs during the partial private key distribution process, with a communication cost of $|G_1|$, where DO represents the data owner. The communication between the TPA and CSs occurs in two processes: the TPA distributes the challenge to CSs and CSs return the proofs as responses, with communication costs of $|\xi(log_2 n + 3|p|)|$ and $\xi(|G_1| + s|p|)$, respectively. The comparison of communication costs between the proposed scheme and IB-MCMC [18], BAPDP-MCMC [21], and CLPDP-MCMS [23] is shown in Table 4.

Computational overhead. The computational costs of the proposed scheme mainly come from three algorithms: **TagGen**, **ProofGen**, and **ProofVer**. The FN needs to generate rn tags with a computational cost of $rn|h| + 3rn|E_{G_1}|$, which is lower than the tag generation cost in IB-MCMC [18], BAPDP-MCMC [21], and CLPDP-MCMS [23]. In order to prove that all copies are stored correctly, the computational cost required for generating proofs for all copy blocks is $r(c + 1)|E_{G_1}|$, which is consistent with their proof generation costs in [21, 23]. Although IB-MCMC [18] only requires rc exponential operations, it is vulnerable to the copy-summation attack. In the integrity verification phase, the computational cost of the proposed scheme is $3Pair + |H| + (s + 1)|E_{G_1}|$, which is independent of the number of copies and challenged blocks, and is significantly lower than the other three schemes, as shown in Table 5.

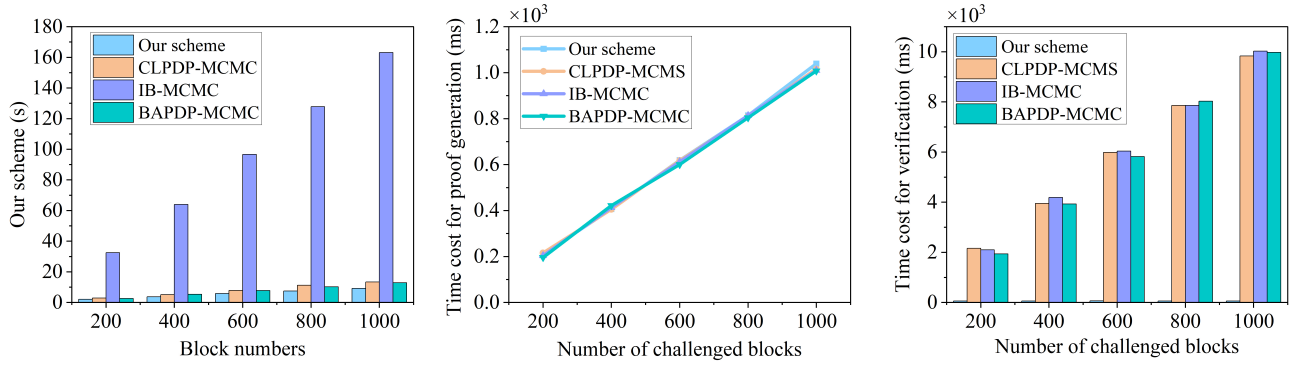
Table 5. Computational overhead.

Schemes	TagGen	ProofGen	Verification
IB-MCMC [18]	$(rn + s)\|H\| + (s + 1)rn\|E_{G_1}\|$	$rc\|E_{G_1}\|$	$3Pair + (rc + s + 1)\|H\| + (rc + s + 2)\|E_{G_1}\|$
CLPDP-MCMS [23]	$rn\|H\| + 2rn\|E_{G_1}\|$	$r(c + 1)\|E_{G_1}\|$	$3Pair + (rc + 1)\|H\| + (rc + s + 1)\|E_{G_1}\|$
BAPDP-MCMC [21]	$rn\|H\| + 2rn\|E_{G_1}\|$	$r(c + 1)\|E_{G_1}\|$	$3Pair + (rc + 1)\|H\| + (rc + s + 1)\|E_{G_1}\|$
Our scheme	$rn\|h\| + 3rn\|E_{G_1}\|$	$r(c + 1)\|E_{G_1}\|$	$3Pair + \|H\| + (s + 1)\|E_{G_1}\|$

6.3. Experimental simulation

To intuitively display the time cost of our proposed scheme, we implement **TagGen**, **ProofGen**, and **ProofVer** and compare them eith IB-MCMC [18], BAPDP-MCMC [21], and CLPDP-MCMS [23]. All code is written in the Python programming language with the Charm-Crypto Library* on an environment running Ubuntu 20.04.3 LTS with 4 GB RAM. The bilinear pairing map is provideded by Type A elliptic curve with a parameter $|p|$ of 160 bits. The simulation results are all averaged across ten tests.

*https://pypi.org/project/charm-crypto/



(a) Time cost for tag generation (b) Time cost for proof generation (c) Time cost for verification

Figure 3. Comparison of time costs (s = 50, r = 3).

We first tested **TagGen**, **ProofGen** as well as **ProofVer**, and the file sizes are 200 KB, 400 KB, 600 KB, 800 KB, and 1 MB, with $s = 50$ and $r = 3$. As shown in Figure 3(a), the time costs for **TagGen** in the above four schemes increase approximately linearly with the number of data blocks. Among them, the proposed scheme has the lowest time cost, with a time variation range of 2.03–9.17 s. This reflects that our scheme is faster in generating tags and has lower latency when uploading data to multiple CSs, making it more efficient for DT data. For the **ProofGen** and **ProofVer**, we tested the time costs when the number of challenged blocks are 200, 400, 600, 800, 1000. As shown in Figure 3(b), our proposed scheme and the schemes in [18, 21, 23] take nearly the same time in generating the proof for one copy. When the number of challenging blocks is 1000, the time required to generate proof for a copy is 1.039 s. In this case, the proposed scheme has a 99% probability of detecting data corruption for a 10 GB file containing 0.46% corrupted data according to Remark 1. In the **ProofVer**, the time cost for verifying the aggregation proof of three copies in our proposed scheme is nearly 60 ms, which is a constant independent of the number of blocks. However, IB-MCMC [18], BAPDP-MCMC [21], and CLPDP-MCMS [23] exhibit a linear upward trend in time costs, as shown in Figure 3(c). This demonstrates that our scheme is computationally TPA-friendly and can provide fast verification for time-sensitive DT data.

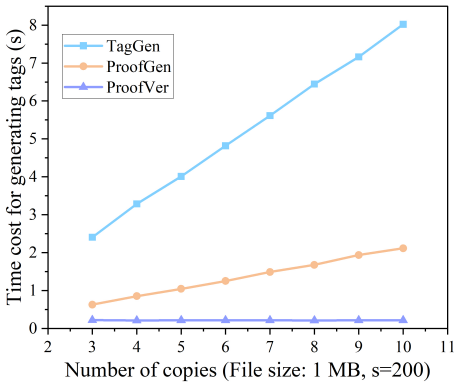


Figure 4. Time cost for generating tags.

Furthermore, we tested the time cost variation of **TagGen**, **ProofGen**, and **ProofVer** in our scheme as the number of copies increases. We set the file size to 1 MB, $s = 200$, and $c = 200$. As shown in Figure 4, the time costs for **TagGen** and **ProofGen** increase linearly with the number of copies, while the time cost to verify the aggregation proof is almost constant and is not affected by the number of copies. This is consistent with the theoretical analysis.

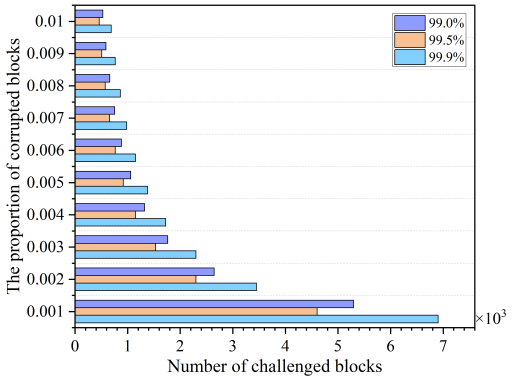


Figure 5. Detection rate

Finally, we simulated the relationship between the corruption rate and the number of challenged blocks when the probability of identifying corrupted files is 99.0%, 99.5%, and 99.9%, respectively. Here we refer to the corruption rate as the ratio of corrupted blocks to total blocks, while the detection rate is the potential of recognizing corrupted blocks in a file. As shown in Figure 5, when the number of challenged blocks is 984, the detection rate of a file with no less than 0.7% corruption rate is as high as 99.9%, the detection rate of a file with at least 0.6% corruption rate is greater than 99.5%, and the detection rate of a file with a corruption rate of less than 0.6% is more than 99.5%.

7. Conclusions

In this paper, we propose an LPDA-MCMC for fog-assisted DT systems. Specifically, built on the end-fog-cloud three-level framework, we design a lightweight integrity verification mechanism based on certificateless technology and fog computing to ensure the integrity and reliability of DT data. Our scheme allows batch auditing of all copies of a file distributed on multiple CSs, and all copies of all files generated within a specific time period, with a constant computational cost. Particularly, we further design a fine-grained error localization mechanism to realize block-level localization of corrupted data. The security analysis demonstrates that our scheme achieves the expected security properties. Performance evaluation confirms that the proposed scheme is efficient, lightweight, and scalable.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant 3072024CFJ2401 and Grant 3072025GH2401.

Conflict of interest

The authors declare that there is no conflict of interest.

References

1. Tao F, Zhang M, Liu Y, Nee AYC, (2018) Digital twin driven prognostics and health management for complex equipment. *CIRP Ann* 67: 169–172. https://doi.org/10.1016/j.cirp.2018.04.055
2. Ren J, Yang P, (2022) Lifecycle data management of nuclear power plant: Framework system and development suggestions. *Strategic Study CAE* 24: 152–159. https://doi.org/10.15302/J-SSCAE-2022.02.012
3. Yang P, Xiong N, Ren J, (2020) Data security and privacy protection for cloud storage: A survey. *IEEE Access* 8: 131723–131740. https://doi.org/10.1109/ACCESS.2020.3009876
4. Yang P, Ren J, (2024) A blockchain-based data auditing scheme with key-exposure resistance for IIoT. *Sci China Inf Sci* 67: 129102. https://doi.org/10.1007/s11432-023-3828-7
5. Chen S, Yang L, Zhao C, Varadarajan V, Wang K, (2022) Double-blockchain assisted secure and anonymous data aggregation for fog-enabled smart grid. *Engineering* 8: 159–169. https://doi.org/10.1016/j.eng.2020.06.018
6. Zhang W, Bai Y, Feng J, (2022) TIIA: A blockchain-enabled threat intelligence integrity audit scheme for IIoT. *Future Gener Comput Syst* 132: 254–265. https://doi.org/10.1016/j.future.2022.02.023
7. Ateniese G, Burns R, Curtmola R, Herring J, Kissner L, Peterson Z, et al. (2007) Provable data possession at untrusted stores. In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 598–609. https://doi.org/10.1145/1315245.1315318
8. Juels A, Kaliski Jr BS, (2007) PORs: Proofs of retrievability for large files. In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 584–597. https://doi.org/10.1145/1315245.1315317
9. Shacham H, Waters B, (2013) Compact proofs of retrievability. *J Cryptology* 26: 442–483. https://doi.org/10.1007/s00145-012-9129-2
10. Shi E, Stefanov E, Papamanthou C, (2013) Practical dynamic proofs of retrievability. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, 325–336. https://doi.org/10.1145/2508859.2516669
11. Erway CC, Küpçü A, Papamanthou C, Tamassia R, (2015) Dynamic provable data possession. *ACM Trans Inf Syst Secur* 17: 1–29. https://doi.org/10.1145/2699909

12. Yang Y, He F, Han S, Liang Y, Cheng Y, (2021) A novel attribute-based encryption approach with integrity verification for CAD assembly models. *Engineering* 7: 787–797. https://doi.org/10.1016/j.eng.2021.03.011

13. Li T, Wang H, He D, Yu J, (2022) Synchronized provable data possession based on blockchain for digital twin. *IEEE Trans Inf Forensics Secur* 17: 472–485. https://doi.org/10.1109/TIFS.2022.3144869

14. Curtmola R, Khan O, Burns R, Ateniese G, (2008) MR-PDP: Multiple-replica provable data possession. In: *Proceedings of the 28th International Conference on Distributed Computing Systems*, 411–420. https://doi.org/10.1109/ICDCS.2008.68

15. Hao Z, Yu N, (2010) A multiple-replica remote data possession checking protocol with public verifiability. In: *Proceedings of the Second International Symposium on Data, Privacy, and E-Commerce*, 84–89. https://doi.org/10.1109/ISDPE.2010.20

16. Liu C, Ranjan R, Yang C, Zhang X, Wang L, Chen J, (2015) MuR-DPA: Top-down levelled multi-replica merkle hash tree based secure public auditing for dynamic big data storage on cloud. *IEEE Trans Comput* 64: 2609–2622. https://doi.org/10.1109/TC.2014.2375190

17. Li J, Yan H, Zhang Y, (2022) Efficient identity-based provable multi-copy data possession in multi-cloud storage. *IEEE Trans Cloud Comput* 10: 356–365. https://doi.org/10.1109/TCC.2019.2929045

18. Chang J, Shao B, Ji Y, Bian G, (2020) Efficient identity-based provable multi-copy data possession in multi-cloud storage, revisited. *IEEE Commun Lett* 24: 2723–2727. https://doi.org/10.1109/LCOMM.2020.3013280

19. Yu H, Yang Z, Waqas M, Tu S, Han Z, Halim Z, et al. (2021) Efficient dynamic multi-replica auditing for the cloud with geographic location. *Future Gener Comput Syst* 125: 285–298. https://doi.org/10.1016/j.future.2021.05.039

20. Li T, Chu J, Hu L, (2023) CIA: A collaborative integrity auditing scheme for cloud data with multi-replica on multi-cloud storage providers. *IEEE Trans Parallel Distrib Syst* 34: 154–162. https://doi.org/10.1109/TPDS.2022.3216614

21. Miao Y, Huang Q, Xiao M, Susilo W, (2022) Blockchain assisted multi-copy provable data possession with faults localization in multi-cloud storage. *IEEE Trans Inf Forensics Secur* 17: 3663–3676. https://doi.org/10.1109/TIFS.2022.3211642

22. Zhou L, Fu A, Yang G, Wang H, Zhang Y, (2022) Efficient certificateless multi-copy integrity auditing scheme supporting data dynamics. *IEEE Trans Dependable Secure Comput* 19: 1118–1132. https://doi.org/10.1109/TDSC.2020.3013927

23. Shen J, Zeng P, Choo KKR, Li C, (2023) A certificateless provable data possession scheme for cloud-based EHRs *IEEE Trans Inf Forensics Secur* 18: 1156–1168. https://doi.org/10.1109/TIFS.2023.3236451

24. Shen J, Chen X, Huang X, Xiang Y, (2023) Public proofs of data replication and retrievability with user-friendly replication. *IEEE Trans Dependable Secure Comput* 21: 2057–2067. https://doi.org/10.1109/TDSC.2023.3299627

25. Sang T, Zeng P, Choo KKR, (2023) Provable multiple-copy integrity auditing scheme for cloud-based IoT. *IEEE Syst J* 17: 224–233. https://doi.org/10.1109/JSYST.2022.3198098

26. Zhang C, Xu Y, Hu Y, Wu J, Ren J, Zhang Y, (2022) A blockchain-based multi-cloud storage data auditing scheme to locate faults. *IEEE Trans Cloud Comput* 10: 2252–2263. https://doi.org/10.1109/TCC.2021.3057771

27. Zhang X, Huang C, Xu C, Zhang Y, Zhang J, Wang H, (2021) Key-leakage resilient encrypted data aggregation with lightweight verification in fog-assisted smart grids. *IEEE Int Things J* 8: 8234–8245. https://doi.org/10.1109/JIOT.2020.3047958

28. Zhang X, Wang X, Gu D, Xue J, Tang W, (2022) Conditional anonymous certificateless public auditing scheme supporting data dynamics for cloud storage systems. *IEEE Trans Network Serv Manage* 19: 5333–5347. https://doi.org/10.1109/TNSM.2022.3189650

29. Su Y, Li Y, Yang B, Ding Y, (2022) Decentralized self-auditing scheme with errors localization for multi-cloud storage. *IEEE Trans Dependable Secure Comput* 19: 2838–2850. https://doi.org/10.1109/TDSC.2021.3075984

30. Wang L, Li Y, Yu Q, Yu Y, (2022) Outsourced data integrity checking with practical key update in edge-cloud resilient networks. *IEEE Wireless Commun* 29: 56–62. https://doi.org/10.1109/MWC.002.2100597

31. Wang B, Li B, Li H, Li F, (2013) Certificateless public auditing for data integrity in the cloud. In: *Proceedings of the 2013 IEEE Conference on Communications and Network Security*, 136–144. https://doi.org/10.1109/CNS.2013.6682701

32. Al-Riyami S, Paterson K, (2003) Certificateless public key cryptography, In: Laih, CS. (eds) *Advances in Cryptology-ASIACRYPT 2003. ASIACRYPT 2003. Lecture Notes in Computer Science*, Berlin: Springer, 452–473. https://doi.org/10.1007/978-3-540-40061-5_29

33. He D, Kumar N, Wang H, Wang L, Choo KKR, (2017) Privacy-preserving certificateless provable data possession scheme for big data storage on cloud. *Appl Math Comput* 314: 31–43. https://doi.org/10.1016/j.amc.2017.07.008

34. Liao Y, Liang Y, Oyewole AW, Nie X, (2019) Security analysis of a certificateless provable data possession scheme in cloud. *IEEE Access* 7: 93259–93263. https://doi.org/10.1109/ACCESS.2019.2928032

35. Ji Y, Shao B, Chang J, Bian G, (2020) Privacy-preserving certificateless provable data possession scheme for big data storage on cloud, revisited. *Appl Math Comput* 386: 125478. https://doi.org/10.1016/j.amc.2020.125478

36. Gudeme J R, Pasupuleti S, Kandukuri R, (2021) Certificateless privacy preserving public auditing for dynamic shared data with group user revocation in cloud storage. *J Parallel Distrib Comput* 156: 163–175. https://doi.org/10.1016/j.jpdc.2021.06.001

37. Li J, Yan H, Zhang Y, (2021) Certificateless public integrity checking of group shared data on cloud storage. *IEEE Trans Serv Comput* 14: 71–81. https://doi.org/10.1109/TSC.2018.2789893

38. Li R, Wang XA, Yang H, Niu K, Tang D, Yang X, (2022) Efficient certificateless public integrity auditing of cloud data with designated verifier for batch audit. *J King Saud Univ Comput Inf Sci* 34: 8079–8089. https://doi.org/10.1016/j.jksuci.2022.07.020

39. Zhao Y, Chang J, (2022) Certificateless public auditing scheme with designated verifier and privacy-preserving property in cloud storage. *Comput Networks* 216: 109270. https://doi.org/10.1016/j.comnet.2022.109270

40. Zhang X, Liu Q, Liu B, Zhang Y, Xue J, (2025) Dynamic certificateless outsourced data auditing mechanism supporting multi-ownership transfer via blockchain systems. *IEEE Trans Network Serv Manage* 22: 2017–2030. https://doi.org/10.1109/TNSM.2025.3525462

41. Barsoum AF, Hasan MA, (2015) Provable multicopy dynamic data possession in cloud computing systems. *IEEE Trans Inf Forensics Secur* 10: 485–497. https://doi.org/10.1109/TIFS.2014.2384391

42. Zhou L, Fu A, Mu Y, Wang H, Yu S, Sun Y, (2021) Multicopy provable data possession scheme supporting data dynamics for cloud-based electronic medical record system. *Inf Sci* 545: 254–276. https://doi.org/10.1016/j.ins.2020.08.031

43. Boneh D, Lynn B, Shacham H, (2004) Short signatures from the Weil pairing. *J Cryptology* 17: 297–319. https://doi.org/10.1007/s00145-004-0314-9

44. Bian G, Chang J, (2020) Certificateless provable data possession protocol for the multiple copies and clouds case. *IEEE Access* 8: 102958–102970. https://doi.org/10.1109/ACCESS.2020.2999208

45. Guo W, Qin S, Gao F, Zhang H, Li W, Jin Z, et al. (2020) Comments on "provable multicopy dynamic data possession in cloud computing systems". *IEEE Trans Inf Forensics Secur* 15: 2584–2586. https://doi.org/10.1109/TIFS.2020.2970591

46. Zhao J, Huang H, He D, Zhang X, Zhang Y, Choo KKR, (2024) IB-IADR: Enabling identity-based integrity auditing and data recovery with fault localization for multicloud storage. *IEEE Int Things J* 11: 27214–27231. https://doi.org/10.1109/JIOT.2024.3398298

Appendix

Proof of Lemma 1

We take three steps to prove this lemma.

Step 1. Given a CDH tuple (g, g^a, g^b), C behaves as a challenger attempting to output the value of g^{ab} by constructing a simulator to solve the solution of the CDH problem. If $\mathcal{A}_1$ forges a valid tag with a non-negligible advantage $\epsilon(\mathcal{A}_1)$ within time $\tau(\mathcal{A}_1)$, the challenger C can construct a simulator that outputs the solution of the CDH instance (g, g^a, g^b) with a non-negligible advantage. Note that $\mathcal{A}_1$ is allowed to replace the public key for any identity of its choice but does not know the value

of the master secret key. We allow $\mathcal{A}_1$ to make a series of queries on identities of its choice, and C should make appropriate responses to these queries. Assuming that all identities queried by $\mathcal{A}_1$ constitute set $\mathcal{D}$. C randomly selects an identity $\widetilde{\mathrm{ID}} \leftarrow \mathcal{D}$ and $m_{\tilde{i}}$ as the challenged identity and data block, respectively.

Setup phase. C initializes the system, sets $\Lambda = g^b$ as the master public key, and publishes the public parameters $(G_1, G_2, H_1, H_2, f_1, f_2, f_3, g, \Lambda)$. Note that C has no idea of the master secret key. For each identity ID, C randomly selects $\rho, \eta, \lambda_j \in Z_p^*$ and computes $u = g^\rho f^\eta, u_j = u^{\lambda_j} \in G_1$.

Queries. C holds five lists L_1, L_2, L_3, L_4, L_5 to record the responses to a series of queries launched by $\mathcal{A}_1$. These lists are initially empty.

- H_1 **queries:** For the identity ID of $\mathcal{A}_1$'s choice, C first checks if there exists the tuple $<$ ID, $w, h >$ in L_1. If not, C randomly selects $w \in Z_p^*$, and computes

$$h = g^w, \quad \text{when} \quad \mathrm{ID} \neq \widetilde{\mathrm{ID}},$$

$$h = \Gamma^w, \quad \text{when} \quad \mathrm{ID} = \widetilde{\mathrm{ID}},$$

  where $\Gamma = g^a$, add $< \mathrm{ID}_i, w, h >$ to L_1. Finally, C responds with h.
- H_2 **queries:** For the input $< \mathrm{CS}, F_n, l, i, t >$, C checks whether the tuple $< \mathrm{CS}, F_n, l, i, t, \nu_i >$ already exists in L_2. If not, C randomly selects $\nu_i \in Z_p^*$ and appends $< \mathrm{CS}, F_n, l, i, t, \nu_i >$ to L_2. Then C sends $\mathcal{A}_1$ ν_i as a reply.
- **Partial Private Key Extraction:** For the request of the partial private key corresponding to ID, C returns Q_{ID} if the tuple $<$ ID, $Q_{\mathrm{ID}} >$ is already in L_3. If not, and $\mathrm{ID} = \widetilde{\mathrm{ID}}$, C aborts (this is denoted as X_1). Otherwise, C gets h from L_1, computes the partial private key as $Q_{\mathrm{ID}} = h^b = \Lambda^w$, add the tuple $<$ ID, $Q_{\mathrm{ID}} >$ into L_3, and sends Q_{ID} to $\mathcal{A}_1$ as a response.
- **Private Key Extraction:** For the input ID, we assume that its public key has not been replaced. C returns sk_{ID} if the list L_4 has the tuple $<$ ID, $sk_{\mathrm{ID}} >$. If not, and $\mathrm{ID} = \widetilde{\mathrm{ID}}$, C aborts (this is denoted as X_2). Otherwise, C randomly selects $\gamma_{ID} \in Z_p^*$ as the secret value and set the private key as $sk_{\mathrm{ID}} = (\gamma_{ID}, Q_{\mathrm{ID}})$. Append $<$ ID, $sk_{\mathrm{ID}} >$ to the list L_4 and sends sk_{ID} to $\mathcal{A}_1$ as a response.
- **Request for Public Key:** C returns the public key if the tuple $<$ ID, $pk_{\mathrm{ID}} >$ exists in the list L_5. For a new ID that has never been queried before, C checks if $\mathrm{ID} = \widetilde{\mathrm{ID}}$. If it holds, C randomly selects $\gamma_{ID} \in Z_p^*$, computes the public key as $pk_{\mathrm{ID}} = (\Lambda^\gamma, g^\gamma)$; otherwise, C gets γ from L_4 and computes $pk_{\mathrm{ID}} = (\Lambda^\gamma, g^\gamma)$. Finally, C appends the tuple $<$ ID, $pk_{\mathrm{ID}} >$ to the list L_5 and returns pk_{ID} as the response.
- **Replace Public Key:** $\mathcal{A}_1$ selects a random number γ' and creates a new $pk'_{\mathrm{ID}} = (\Lambda^{\gamma'}, g^{\gamma'})$ to replace the current public key value pk_{ID} for identity ID. C aborts if $\mathrm{ID} = \widetilde{\mathrm{ID}}$ (this is denoted as X_3). If not, C updates $<$ ID, $pk_{\mathrm{ID}} >$ to $<$ ID, $pk'_{\mathrm{ID}} >$.
- **Tag queries:** For the input $<$ ID, $\mathrm{CS}\|F_{name}\|l\|i\|t, T_i >$, $\mathrm{ID} \neq \widetilde{\mathrm{ID}}$, C computes the tag by using the private key. Otherwise, for $i \neq \tilde{i}$, C randomly selects $\rho_i \in Z_p^*$, sets $H_1(\widetilde{\mathrm{ID}})^{H_2(\mathrm{CS}\|F_{name}\|l\|i\|t)} = g^{\rho_i}$, and computes the tag as

$$T_{l,i} = (H_1(\widetilde{\mathrm{ID}})^{bH_2(CS\|F_{name}\|l\|i\|t)} \cdot \prod_{j=1}^{s} u_j^{m_{l,i,j}})^{\gamma_{\widetilde{\mathrm{ID}}}} = (\Lambda^{\gamma_{\widetilde{\mathrm{ID}}}\rho_i} \cdot (u^{\gamma_{\widetilde{\mathrm{ID}}}})^{\sum_{j=1}^{s} \lambda_j m_{l,i,j}}),$$

  which satisfies Eq (A.1). If $\mathrm{ID} = \widetilde{\mathrm{ID}}$ and $i = \tilde{i}$, C aborts (this is denoted as X_4).

Challenge phase: C requires $\mathcal{A}_1$ to generate the tag for data block $m_{\tilde{i}}$ with regard to the identity $\widetilde{\mathrm{ID}}$ and public key $pk_{\widetilde{\mathrm{ID}}}$, which is never queried before in tag query oracle.

Forgery phase. $\mathcal{A}_1$ forges a tag $\tilde{T}$ satisfying Eq (A.1) and returns $\tilde{T}$ as a reply.

To solve the CDH problem, C first recovers $h = \Gamma^w$, $v_{\tilde{i}}$, $pk_{\widetilde{\mathrm{ID}}} = (\Lambda^{\gamma_{\widetilde{\mathrm{ID}}}}, g^{\gamma_{\widetilde{\mathrm{ID}}}})$ from L_1, L_2, L_5, respectively. If the forgery $\tilde{T}$ is valid, it satisfies Eq (A.1).

$$\begin{aligned} e(g,\tilde{T}) &= e(\Lambda^{\gamma_{\widetilde{\mathrm{ID}}}}, h^{v_{\tilde{i}}}) \cdot e(g^{\gamma_{\widetilde{\mathrm{ID}}}}, \prod_{j=1}^{s} u_j^{m_{\tilde{i},j}}) \\ &= e(g^{b\gamma_{\widetilde{\mathrm{ID}}}}, g^{awv_{\tilde{i}}}) \cdot e(g^{\gamma_{\widetilde{\mathrm{ID}}}}, \prod_{j=1}^{s} u^{\lambda_j m_{\tilde{i},j}}) \\ &= e(g, g^{ab\gamma_{\widetilde{\mathrm{ID}}} w v_{\tilde{i}}}) \cdot e(g, u^{\gamma_{\widetilde{\mathrm{ID}}} \sum_{j=1}^{s} \lambda_j m_{\tilde{i},j}}). \end{aligned}$$

It is easy to derive that

$$g^{ab} = (\tilde{T}/u^{\gamma_{\widetilde{\mathrm{ID}}} \sum_{j=1}^{s} \lambda_j m_{\tilde{i},j}})^{(\gamma_{\widetilde{\mathrm{ID}}} w v_{\tilde{i}})^{-1}}. \tag{A.1}$$

Step 2. We analyze the probability of solving the CDH problem. A valid forgery requires the simulation process to be completed without interruption. The events that the simulation may abort include:

(1) The events X_1, X_2, X_3, X_4.

(2) $\mathrm{ID} \neq \widetilde{\mathrm{ID}}$ in the forgery phase, which is denoted as X_5.

Notice that $Pr[\neg X_1] = (1 - \frac{1}{|\mathcal{D}|})^{q_1}$, $Pr[\neg X_2|\neg X_1] = (1 - \frac{1}{|\mathcal{D}|})^{q_2}$, $Pr[\neg X_3|\neg X_1 \wedge \neg X_2] = (1 - \frac{1}{|\mathcal{D}|})^{q_3}$, $Pr[\neg X_4|\neg X_1 \wedge \neg X_2 \wedge \neg X_3] = (1 - \frac{1}{|\mathcal{D}|})^{q_4}$, $Pr[\neg X_5|\neg X_1 \wedge \neg X_2 \wedge \neg X_3 \wedge \neg X_4] = \frac{1}{|\mathcal{D}|}$. Let q_1, q_2, q_3, q_4 respectively denote the number of requests for extracting partial private key, extracting private key, replacing public key, and querying tag, respectively. Then the probability that the simulator does not abort is

$$Pr[\neg X_1 \wedge \neg X_2 \wedge \neg X_3 \wedge \neg X_4 \wedge \neg X_5] \geq (1 - \frac{1}{|\mathcal{D}|})^{q_1+q_2+q_3+q_4} \times \frac{1}{|\mathcal{D}|} \geq \frac{(1-|\mathcal{D}|)^q}{|\mathcal{D}|^{q+1}},$$

where $q = q_1 + q_2 + q_3 + q_4$. Note that $Pr[Adv_{\mathcal{A}_1}] = \epsilon(\mathcal{A}_1)$ is a non-negligible value. Therefore, the probability of solving the CDH problem is at least $\epsilon_1 = \epsilon(\mathcal{A}_1)\frac{(1-|\mathcal{D}|)^q}{|\mathcal{D}|^{q+1}}$, which is non-negligible.

Step 3. We analyze the time cost of solving the CDH problem. Set t_0 as the total time for recovering $h, v_{\tilde{i}}, pk_{\widetilde{\mathrm{ID}}}, t_{sz}, t_{mz}, t_{inz}$ as the time consumed by an addition, a multiplication, and an inverse operation in Z_p^*, t_{e1}, t_{m1} as the time consumed by an exponentiation and an multiplication in G_1, respectively. From Eq (A.1), the total time required is

$$t_1 = \tau(\mathcal{A}_1) + t_0 + 2t_{e1} + t_{m1} + (s-1)t_{ms} + (s+4)t_{mz} + t_{inz},$$

which is bounded.

That is to say, the probability of successfully solving the CDH problem in a bounded time is not insignificant if $\mathcal{A}_1$ forges a valid tag with a non-negligible advantage $\epsilon(\mathcal{A}_1)$ within time $\tau(\mathcal{A}_1)$. However, this contradicts the CDH assumption.

This completes the proof.

Proof of Lemma 2

Given a DL tuple $(g, f = g^{\tilde{r}})$, we show that if $\mathcal{A}_1$ has a non-negligible advantage in forging a valid proof, the probability of solving the DL problem is non-negligible. If the forged proof $P^* = (\sigma^*, \{\mu_j^*\}_{1\le j\le s})$ passes the verification, there is

$$e(g, \sigma^*) = e(pk_1, H_1(ID)^{\sum_{l=1}^{r}\sum_{i\in I} h_{l,i} v_i a_l}) \cdot e(pk_2, \prod_{j=1}^{s} u_j^{\mu_j^*}). \tag{A.2}$$

Let the correct proof be $P = (\sigma, \{\mu_j\}_{1\le j\le s})$. Obviously, it satisfies Eq (A.3). It is impossible for a Type I adversary ($\mathcal{A}_1$) to forge a valid aggregate tag $\sigma^* \neq \sigma$, since we have proved that a Type I adversary ($\mathcal{A}_1$) cannot generate a valid tag with regard to $\widetilde{\mathrm{ID}}$ for any specified block $\tilde{m}$. So we get

$$\prod_{j=1}^{s} u_j^{\mu_j^*} = \prod_{j=1}^{s} u_j^{\mu_j}, \tag{A.3}$$

where $u_j = u^{\lambda_j}$. Then the following three equations are equivalent to Eq (A.3).

$$(g^{\rho} f^{\eta})^{\sum_{j=1}^{s} \lambda_j \mu_j^*} = (g^{\rho} f^{\eta})^{\sum_{j=1}^{s} \lambda_j \mu_j},$$

$$f^{\eta \sum_{j=1}^{s} \lambda_j (\mu_j^* - \mu_j)} = g^{-\rho \sum_{j=1}^{s} \lambda_j (\mu_j^* - \mu_j)},$$

$$f = g^{-\rho \sum_{j=1}^{s} \lambda_j (\mu_j^* - \mu_j) / (\eta \sum_{j=1}^{s} \lambda_j (\mu_j^* - \mu_j))}.$$

Thus we output $\tilde{r} = -\rho \sum_{j=1}^{s} \lambda_j (\mu_j^* - \mu_j) / (\eta \sum_{j=1}^{s} \lambda_j (\mu_j^* - \mu_j))$ for the DL tuple $(g, f = g^{\tilde{r}})$, and this contradicts the DL assumption.

This completes the proof.

Proof of Lemma 3

If $\mathcal{A}_2$ forges a valid tag with a non-negligible advantage $\epsilon(\mathcal{A}_2)$ within time $\tau(\mathcal{A}_2)$, challenger C can create a simulator that outputs the solution of the CDH instance (g, g^a, g^b) with a non-negligible advantage. Specifical analysis as follows.

Setup. C first initializes the master secret key ζ and reveals it to the adversary $\mathcal{A}_2$. Publish the public parameters $(G_1, G_2, g, Y = g^{\zeta})$, $\Gamma = g^a$, $\Lambda = g^b$. Suppose that all identities queried by $\mathcal{A}_2$ constitute set $\mathcal{D}$. Randomly select $\widetilde{\mathrm{ID}} \in \mathcal{D}$ and $m_{\tilde{i}}$ as the challenged identity and data block. For each $\mathrm{ID} \neq \widetilde{\mathrm{ID}}$, randomly choose $\kappa, \lambda_j \in Z_p^*$, and compute $u = g^{\kappa}, u_j = u^{\lambda_j}, j = 1, \cdots, s$; for $\widetilde{\mathrm{ID}}$, choose $\kappa, \beta, \lambda_j \in Z_p^*$, and compute $u = \Lambda^{\kappa} g^{\beta}, u_j = u^{\lambda_j}, j = 1, \cdots, s$.

Queries. We allow $\mathcal{A}_2$ to make a series of queries about identities on its choice. C should make appropriate responses to these queries and record them into lists S_1, S_2, S_3, S_4, S_5, which are initially empty.

- H_1 **queries:** C responds to the request of $\mathcal{A}_2$ as H_1 query in **Step 1** of Lemma 1.
- H_2 **queries:** For the input $< \mathrm{CS}, F_n, l, i, t >$, C returns ν_i if the tuple $< \mathrm{CS}, F_n, l, i, t, \nu_i >$ already exists in S_2. If not, and $i = \tilde{i}$, randomly select $\nu_i \in Z_p^*$ and append $< \mathrm{CS}, F_n, l, i, t, \nu_i >$ to L_2. Otherwise, set $\nu_i = -(\kappa \sum_{j=1}^{s} \lambda_j m_{l,i,j})(w\zeta)^{-1}$. C sends ν_i to $\mathcal{A}_2$.

- **Private Key Extraction:** For the input ID, C returns sk_{ID} if the list S_3 has the tuple < ID, sk_{ID} >. If not, and ID = $\widetilde{\mathrm{ID}}$, it aborts. Otherwise randomly select $\gamma \in Z_p^*$ as the secret value and set the private key as $sk_{\mathrm{ID}} = (\gamma, Q_{\mathrm{ID}})$. Append < ID, sk_{ID} > into the list S_3 and respond with sk_{ID} to $\mathcal{A}_2$.
- **Request for Public Key:** C returns the public key if the tuple < ID, pk_{ID} > exists in the list S_4. For the new ID never queried before, C checks if ID = $\widetilde{\mathrm{ID}}$. If it holds, compute the public key as $pk_{\mathrm{ID}} = (\Lambda^{\zeta}, \Lambda)$; otherwise, get γ by running **KeyGen** and compute $pk_{\mathrm{ID}} = (Y^{\gamma}, g^{\gamma})$. Finally, C appends the tuple < ID, pk_{ID} > to the list S_4 and returns pk_{ID} as the response.
- **Tag queries:** If the tuple < ID, CS$\|F_{name}\|l\|i\|t, T_{l,i}$ > exists in the list S_4, C returns the tag. If not and ID = $\widetilde{\mathrm{ID}}$, $i = \tilde{i}$, it aborts. Otherwise, compute the tag via using the private key when ID $\neq \widetilde{\mathrm{ID}}$; compute the tag as follows when ID = $\widetilde{\mathrm{ID}}$,

$$\begin{aligned} T_{l,i} &= (H_1(\mathrm{ID}^{\zeta\nu_i}) \prod_{j=1}^{s} u_j^{m_{l,i,j}})^a \\ &= g^{abw\zeta\nu_i} \cdot g^{a(b\kappa+\beta)\sum_{j=1}^{s}\lambda_j m_{l,i,j}} \\ &= g^{ab(w\zeta\nu_i+\kappa\sum_{j=1}^{s}\lambda_j m_{l,i,j})} \cdot \Lambda^{\beta\sum_{j=1}^{s}\lambda_j m_{l,i,j}} \\ &= \Lambda^{\beta\sum_{j=1}^{s}\lambda_j m_{l,i,j}}. \end{aligned}$$

Challenge phase. C requires $\mathcal{A}_2$ to generate a valid tag for data block $m_{\tilde{i}}$ for the identity $\widetilde{\mathrm{ID}}$ and public key $pk_{\widetilde{\mathrm{ID}}}$, and this tag generation has never been queried before.

Forgery phase. C first recovers $h = \Gamma^{\omega}, \nu_{\tilde{i}}, pk_{\widetilde{\mathrm{ID}}} = (\Lambda^{\zeta}, \Lambda)$ from S_1, S_2, S_4, respectively. If the forgery $\tilde{T}$ is valid, it satisfies Eq (A.2).

$$\begin{aligned} e(g, \tilde{T}) &= e(\Lambda^{\zeta}, h^{\nu_{\tilde{i}}}) \cdot e(\Lambda, \prod_{j=1}^{s} u_j^{m_{\tilde{i},j}}) \\ &= e(g^{a\zeta}, g^{b\omega\nu_{\tilde{i}}}) \cdot e(\Lambda, \prod_{j=1}^{s} g^{(b\kappa+\beta)\lambda_j m_{\tilde{i},j}}) \\ &= e(g, g^{ab\zeta\omega\nu_{\tilde{i}}}) \cdot e(g, \Lambda^{(b\kappa+\beta)\sum_{j=1}^{s}\lambda_j m_{\tilde{i},j}}) \end{aligned}$$

It is easy to derive that $g^{ab} = (\tilde{T}/\Lambda^{(b\kappa+\beta)\sum_{j=1}^{s}\lambda_j m_{\tilde{i},j}})^{(\zeta\omega\nu_{\tilde{i}})^{-1}}$.

Similar to Steps 2 and 3 in the proof of Lemma 1, it is easy to prove that if a valid tag is forged with advantage $\epsilon(\mathcal{A}_2)$ within time $\tau(\mathcal{A}_2)$, C can solve the CDH problem with a non-negligible advantage $\epsilon_2 \geq \epsilon(\mathcal{A}_2)\frac{(|\mathcal{D}|-1)^{q_s+q_a}}{|\mathcal{D}|^{q_s+q_a+1}}$ within $t_2 \leq \tau(\mathcal{A}_2) + t_0 + t_{m1} + 2t_{e1} + st_{sz} + (s+4)t_{mz} + t_{inz}$, where q_s, q_a denote the number of queries for private key and tag respectively. This contradicts the CDH assumption.

This completes the proof.

Proof of Theorem 2

For the challenge index set I, parameters $\{\nu_i\}_{i\in I}$, and $\{a_l\}_{l\in R_k}$, CS$_k$ computes

$$\sigma_k = \prod_{l\in R_k}(\prod_{i\in I} T_{l,i}^{\nu_i})^{a_l},$$

$$\mu_{j,k} = \sum_{l\in R_k} a_l \sum_{i\in I} m_{l,i,j}\nu_i, \quad j = 1, 2, \cdots, s.$$

The existence of parameters $\{a_l\}_{l\in R_k}$ ensures that every copy of the challenged blocks stored on CS_k must be computed during the proof generation phase.

Proof of Theorem 3

Without loss of generality, we set the malicious CS to CS_k, which attempts to pass current verification using valid proof $P'_k = (\sigma'_k, \{\mu'_{j,k}\}_{j=1}^{s})$ corresponding to a previous challenge with parameters $I', \{v'_i\}_{i\in I'}$. If the current verification is passed, Eq (A.4) holds. It is clear that Eq (A.5) holds naturally.

$$e(g, \sigma'_k) = e(pk_1, H_1(ID)^{\sum_{l\in R_k}\sum_{i\in I} h_{l,i}a_l v_i}) \cdot e(pk_2, \prod_{j=1}^{s} u_j^{\mu'_j}), \tag{A.4}$$

$$e(g, \sigma'_k) = e(pk_1, H_1(ID)^{\sum_{l\in R_k}\sum_{i\in I'} h'_{l,i}a'_l v'_i}) \cdot e(pk_2, \prod_{j=1}^{s} u_j^{\mu'_j}), \tag{A.5}$$

where I and $\{v_i\}_{i\in I}$ are the current challenge index set and parameters, respectively. From the two equations above, we derive

$$\sum_{l\in R_k}\sum_{i\in I} h_{l,i}a_l v_i = \sum_{l\in R_k}\sum_{i\in I'} h'_{l,i}a'_l v'_i. \tag{A.6}$$

Since the challenge index I ($|I| = c$) and parameters $\{a_l\}_{l\ R_k}, \{v_i\}_{i\in I}$ are closely related to the times-tamp of current challenge initiation, there must be $I \neq I'$, $\{a_l\}_{R_k} \neq \{a'_l\}_{R_k}$, $\{v_i\}_{i\in I} \neq \{v'_i\}_{i\in I'}$, $\{h_{l,i}\}_{l\in R_k, i\in I} \neq \{h'_{l,i}\}_{l\in R_k, i\in I'}$. Therefore, the probability that Eq (A.6) holds is $(1/p)^{c|R_k|+|R_k|+c-1}$, which is negligible.

Proof of Theorem 4

This proof consists two parts.

We first demonstrate that any malicious CS_k cannot pass TPA's verification by replacing the challenged index $l_0, i_0 \in I$ with another $l', i' \neq I$. Let the correct final proof be $P = (\sigma = \prod_{l=1}^{r}\prod_{i\in I} T_{l,i}^{v_i})$, which satisfies Eq (A.3). If the final proof generated after l_0, i_0 is replaced with l, i' satisfies Eq (A.3), then the following relationship can be derived:

$$T_{l',i'}/T_{l,i} = \prod_{j=1}^{s} u_j^{\gamma(m_{l',i',j'} - m_{l,i,j})}. \tag{A.7}$$

While from the perspective of tag generation, there is

$$T_{l',i'}/T_{l,i} = H_1(\mathrm{ID})^{\alpha\gamma(h_{l',i'} - h_{l,i})} \prod_{j=1}^{s} u_j^{\gamma(m_{l',i',j'} - m_{l,i,j})}. \tag{A.8}$$

Divide Eq (A.7) by Eq (A.8), we get $h_{l',i'} = h_{l,i}(mod p)$. This contradicts the collision resistance of hash function H_2.

Next, we prove that any malicious CS cannot use a valid partial proof generated by another CS as its own to pass the TPA's verification. Without loss of generality, consider CS_2 eavesdropping on

CS_3's partial proof $(\sigma_3, \{\mu_{j,3}\}_{1\le j\le s})$ and sending it to the TPA as its own partial proof. If the verification passes, there are

$$e(g,\sigma_3) = e(pk_1, H_1(ID)^{\sum_{l\in R_2}\sum_{i\in I} h_{l,i}a_l v_i}) \cdot e(pk_2, \prod_{j=1}^{s} u_j^{\mu_{j,3}}).$$

The equation below naturally holds for $(\sigma_3, \{\mu_{j,3}\}_{1\le j\le s})$.

$$e(g,\sigma_3) = e(pk_1, H_1(ID)^{\sum_{l\in R_3}\sum_{i\in I} h_{l,i}a_l v_i}) \cdot e(pk_2, \prod_{j=1}^{s} u_j^{\mu_{j,3}}).$$

Easy to get

$$\sum_{l\in R_2}\sum_{i\in I} h_{l,i}a_l v_i = \sum_{l\in R_3}\sum_{i\in I} h_{l,i}a_l v_i. \tag{A.9}$$

Note that the hash values $h_{l,i}, l \in R_2$ are not equal to $h_{l,i}, l \in R_3$, and $a_l, l \in R_2$ are not equal to $a_l, l \in R_3$, since the copy index set $R_2 \cap R_3 = \emptyset$. Thus the probability that Eq (A.9) holds is less than $(1/p)^{c|R_3|+|R_3|}$, which is negligible. Λ