SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348-8549/ https://doi.org/10.14445/23488549/IJECE-V13I11P117

Volume 13 Issue 11, 279-299, November 2026
© 2026 Seventh Sense Research Group®

Original Article

A Lightweight Hierarchical Merkle-based Integrity Verification Framework for Resource-Constrained Edge Devices in Cloud Environments

N. Musrat Sultana[1,2*], K. Srinivas[1]

[1]*Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Hyderabad, Telangana, India.*
[2]*Department of Computer Science and Engineering, Mahatma Gandhi Institute of Technology, Hyderabad, Telangana, India.*

[*]*Corresponding Author: sultanamgit@klh.edu.in*

Received: 22 July 2026 Revised: 31 October 2026 Accepted: 05 November 2026 Published: 27 November 2026

Abstract *- The growth of cloud computing and edge and Internet-of-Things (IoT) environments has led to the increased need to find efficient mechanisms that guarantee the integrity of the outsourced data. Most contemporary systems produce vast amounts of information using resource-limited edge devices, which have limited memory, processing power and power resources. Conventional integrity verification methods like Provable Data Possession and Proof of Retrievability rely on computationally costly cryptographic functions and are therefore inappropriate for lightweight edge infrastructures. To address this issue, this paper presents a lightweight, cloud-integrated, edge-specific hierarchical data integrity verification model. The scheme proposed is lightweight homomorphic hashing combined with hierarchical aggregation of a Merkle hash tree and batch auditing to alleviate the burden of computation on edge devices. Under this architecture, edge nodes perform simple hash computations, edge gateways handle aggregation and integrity coordination, and then connect to the cloud storage server. The block hash and the Merkle commitment are formally defined; a ring-SIS aggregate is included and, contingent on well-stated assumptions, accepted proofs are bound to the committed data, with corruption being recognised with quantified probability, and tampered tree nodes and a gateway-server collusion on the presented state are also detected. We simulated the framework in the cloud-edge using synthetic edge workloads and ToN-IoT telemetry data. The results of the experiments show that in the simulated environment, the proposed model is more efficient than the traditional PDP and PoR methods. Compared with these baselines, the framework saves 46-59% of instructions, 50-60% of communication, 40-67% of storage overhead, and approximately 47-58% of energy per audit, while enabling tampering detection within the sampling bound derived in the security analysis. These results indicate that hierarchical integrity verification offers a feasible and scalable mechanism for ensuring the security of data produced within large-scale IoT and edge-cloud systems.*

Keywords *- Cloud computing, Edge computing, Data integrity verification, Merkle hash tree, Lightweight cryptography, Proof of data possession, Proof of Retrievability.*

1. Introduction

Cloud computing has turned out to be a cornerstone paradigm of providing scalable computing services and storage services based on distributed infrastructures. Cloud platform applications, data processing, and digital service management are increasingly becoming a part of organisations in all industries since they offer benefits like elasticity, cost-effectiveness, and high availability. Armbrust et al. [1] state that cloud computing allows on-demand access to shared computing resources that can be brought online quickly with minimal management overhead. With the increasing pace of the digital transformation in the industry (healthcare, finance, smart cities, industrial automation, etc.), the amount of information kept in cloud-based systems continues to expand rapidly [2, 3]. Nonetheless, outsourcing data to remote Cloud Service Providers (CSPs) is associated with a high level of security issues connected with trust, transparency, and data safety. These obstacles include the security of outsourced information, and this has emerged as a major need for a credible cloud computing system.

Nowadays, with edge computing and Internet of Things (IoT) technologies that have been introduced in the recent past, the traditional cloud architecture has continued to evolve into a distributed cloud-edge ecosystem. This introduces edge devices in large quantities in these environments, such as sensors, embedded systems, and wearable devices that generate and preprocess data and send it to cloud servers to be stored and analyzed [4]. The edge devices are usually equipped with limited computational resources, reduced memory and communication bandwidth, as well as limited energy. Most IoT devices are based on low-frequency processors and have little memory, which is

 This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

not suitable for conducting complex cryptographic tasks [5]. Moreover, energy usage directly influences the operational range of the battery-powered edge equipment used in remote/mission-critical locations. Consequently, whereas cloud infrastructures present a high computational potential, the machines that generate data do not always have the capacity to perform intensive security functions.

Cloud-based information security can be generally defined in terms of three key principles: confidentiality, integrity, and availability. Confidentiality guarantees that data are not compromised by unauthorised users due to encryption processes, whereas integrity guarantees that the data stored or being transferred are correct and free of any modification during the course of their lifecycle [6]. Despite the fact that encryption algorithms like the Advanced Encryption Standard (AES) and Elliptic Curve Cryptography (ECC) are effective in ensuring that the information is not unlawfully disclosed, it is not inherent in the algorithm that cloud servers keep data in the proper format and eliminate the information without alteration or erasure. Data may be corrupted or manipulated due to hardware failures, rogue insiders, configuration mistakes or hacked servers. This means that encryption does not guarantee trust in outsourced storage systems, and specific mechanisms of data integrity verification are needed.

To curb integrity verification of outsourced storage systems, a number of cryptographic protocols have been put forward. PDP, which was proposed by Ateniese et al. [7], enables a client to confirm that the original data is held by a remote server without downloading the whole dataset. Likewise, Proof of Retrievability (PoR) is an approach introduced by Juels and Kaliski [8] and expanded by Shacham and Waters [9], which guarantees that the data stored in the cloud may be completely retrieved, even when it is partially damaged. These strategies are based on challenge-response strategies, homomorphic authenticators and cryptographic verification labels to authenticate the integrity of data. Although the theoretical guarantees of PDP and PoR schemes are very robust, they can be associated with computationally intensive algorithms like modular exponentiation, pairing-based cryptography or large metadata management, which can play a crucial role in adding computational overhead to clients [10, 11]. The computational cost of traditional integrity verification protocols is a serious problem for resource-constrained edge environments. In the cloud-integrated IoT, the owner of the data is often an embedded processor or microcontroller with minimal processing power. The repetition of integrity checking with heavy primitives of cryptography can lead to high rates of energy usage and communication. In addition, a number of integrity verification models have employed the use of Third-Party Auditors (TPAs) to complete verification activities on behalf of the data owner. Though this design is more computationally efficient for the clients, it increases trust assumptions and could create possible privacy concerns [12]. Regular auditing of such environments with a highly dynamic edge is not efficient under a traditional scheme where the data are continuously generated.

The other significant drawback of standard methods of integrity verification is connected with the scale. With increased volume of outsourced data, there is also an increase in the cost of computing and communication involved in the verification. Despite the introduction of batch auditing methods to minimize the verification overhead, several available solutions are based on complicated cryptographic primitives incompatible with lightweight devices [13]. Also, existing frameworks do not take into account the fact that cloud-edge architecture is hierarchical. Practically, intermediate edge gateways will usually be able to compute more than single sensor nodes, and may also aggregate verification data before transmitting to cloud servers. Nevertheless, there are several current models of integrity verification that do not take advantage of this hierarchical form to their benefit.

Recent studies have thus aimed at coming up with lightweight integrity verification mechanisms that can be used within resource-constrained environments. A number of lightweight cryptography tools and hash-based verification models have been considered to decrease the computational complexity at the expense of a reasonable security guarantee [14]. Merkle hash tree structures have also been implemented in distributed systems to offer efficient integrity checks with logarithmically sized proofs [15]. Merkle trees enable the efficient integrity verification of data blocks without having to re-read the entire data block by organising the data blocks in a hierarchical tree structure [16]. This type of hierarchical verification would be compatible with cloud-edge architectures, in which intermediate nodes have the option to aggregate verification information and then communicate with centralised cloud servers.

Despite advances in cloud data integrity verification, several challenges remain in applying existing approaches to resource-constrained edge–cloud environments. First, conventional Provable Data Possession (PDP) and Proof of Retrievability (PoR) schemes may impose considerable computational, communication, or metadata-management overhead, depending on their cryptographic constructions and auditing configurations. These costs can be problematic for edge devices with limited processing capacity, memory, bandwidth, and energy resources. Second, lightweight hash-based verification can reduce computational requirements, but its effectiveness depends on the underlying authentication mechanism and the protection of the reference integrity information. A hash value alone does not establish the completeness or retrievability of remotely stored data. Third, existing verification approaches do not always exploit the different computational capabilities of edge devices, intermediate gateways, and cloud servers through a coordinated hierarchical design. Consequently, there is a need to investigate an integrity verification framework that combines lightweight block-level hashing, hierarchical Merkle-tree authentication, and batch auditing while explicitly evaluating computational cost, communication overhead, metadata storage, and energy consumption. The central research problem addressed in

this study is whether such a hierarchical design can reduce verification overhead in simulated cloud–edge environments without compromising the integrity guarantees provided by its underlying cryptographic mechanisms.

Due to these issues, the current study will offer a lightweight hierarchical data integrity verification system to be used in cloud-integrated systems that include resource-constrained edge devices. The suggested structure is a combination of lightweight homomorphic hashing, hierarchical aggregation with a Merkle tree and batch auditing systems to reduce the computational cost and, at the same time, provide high integrity assurance. With edge gateway offloading of computationally demanding functions and retaining end-to-end integrity checking, the proposed model will offer scalable and efficient integrity checking that can be applicable to the current cloud-edge systems.

1.1. Novelty and Research Contribution

This study's novelty lies in integrating hierarchical Merkle-tree authentication, lightweight block-level integrity processing, and batch-auditing coordination within a cloud–edge architecture designed for resource-constrained devices. Unlike conventional PDP and PoR approaches, which employ scheme-dependent authentication mechanisms and may introduce additional computational, communication, or storage costs, the proposed framework emphasizes the distribution of verification responsibilities across edge devices, gateways, cloud storage, and the verifier. In contrast to approaches that primarily focus on lightweight hashing or Merkle-tree authentication in isolation, this study investigates their coordinated use in a hierarchical verification workflow and evaluates its performance across proof-generation time, verification time, communication overhead, metadata storage, and estimated energy consumption. The contribution is therefore architectural and experimental: it examines whether this combination can reduce verification overhead under specified cloud–edge simulation conditions. The claimed novelty does not imply that Merkle trees, lightweight hashing, or batch auditing are individually new; rather, it concerns their integration and evaluation within the proposed framework. The technical validity of this contribution depends on the precise hash construction, authentication protocol, and security assumptions defined in Section 3.

The main contributions of this work are summarised as follows:

- Hierarchical verification architecture: A cloud–edge integrity verification architecture that allocates data generation, integrity processing, aggregation, and verification tasks according to the roles of edge devices, gateways, cloud storage, and the verifier.
- Merkle-based batch verification: A hierarchical Merkle-tree authentication and batch-auditing design intended to reduce the need to transmit individual authentication information for every challenged block, subject to the verification protocol defined in the methodology.
- Performance and complexity evaluation: An evaluation of proof-generation time, verification time, communication overhead, metadata storage, and estimated energy consumption, together with a complexity analysis under explicitly stated assumptions.
- Comparative experimental assessment: A comparison with PDP and PoR baselines using synthetic edge workloads and ToN-IoT telemetry data in the specified simulated cloud–edge environment, with the aim of examining performance, scalability, and implementation limitations.

1.1.1. Formal Security Analysis

Definition of the additive block hash function and the Merkle commitment, with results on binding of leaves, verification soundness, probability of sampling-based detection, resistance to replay and rollback, aggregation of binding and resistance to gateway collusion (Section 3.6).

Experimental Validation

The advantages in performance are validated through experimental tests using (ToN-IoT) realistic IoT telemetry or synthetic workloads on a simulated cloud–edge infrastructure.

1.2. Research Questions

The research questions discussed in this study are as follows:

RQ1: How can integrity verification operations be distributed across edge devices, gateways, and cloud servers to accommodate the resource constraints of edge devices?

RQ2: To what extent does hierarchical Merkle-tree authentication with batch auditing reduce proof-generation time, verification time, and communication overhead compared with the selected PDP and PoR baselines?

RQ3: How does the proposed framework affect metadata storage and estimated energy consumption as the number of data blocks increases, and how does its performance scale under the specified cloud–edge simulation conditions?

1.3. Structure of the Paper

The rest of the paper is arranged in the following way. Section 2 provides a literature review on related work on data integrity verification of clouds, such as Provable Data Possession, Proof of Retrievability, and lightweight cryptographic verification mechanisms. In Section 3, the suggested lightweight hierarchical integrity verification scheme is provided, and the system architecture, hashing model, Merkle tree aggregation scheme, audit protocol, and formal security and complexity analysis are discussed. Section 4 explains the experimental arrangement, data generation procedure and the simulation platform applied in the assessment. In Section 5, the proposed approach is compared with the current integrity verification schemes by providing the performance evaluation results. Lastly, Section 6 wraps up the research and provides limitations and future research suggestions.

2. Literature Review

2.1. Cloud Data Integrity Checking

Cloud computing has also had a great impact on the area of data storage in that organizations are able to outsource huge amounts of data to remote infrastructure. Nevertheless, this move poses severe security issues since users will not have direct control of their data anymore. Maintaining accuracy, completeness and integrity of information in the cloud has thus been of significant research interest in cloud security.

Initial research stressed the necessity to have mechanisms that would help to establish the accuracy of remotely stored information without necessarily retrieving the entire data from the cloud [7, 8]. As emphasised in the survey of Subashini and Kavitha [6], the distance between data ownership and data management in cloud infrastructures creates vulnerabilities that include unauthorised modifications, insider attacks, as well as accidental data corruption.

Earlier data integrity checking methods were based on the use of hash functions or digital signatures. In these techniques, the data owner will calculate a hash of each block of data and outsource it to the cloud and then the data owner will recalculate the hash on retrieval to ensure the data has been correctly sent. These techniques are conceptually straightforward, but they have the disadvantage of needing the entire dataset to be downloaded in order to be verified; thus, they are highly scaled down by communication overheads and inability to scale well with large datasets [7]. Therefore, cloud environments that have been increasing data volumes at high rates need more efficient remote integrity verification mechanisms.

2.2. Provable Data Possession (PDP)

The idea of Provable Data Possession (PDP) schemes emerged as a cryptographic tool used to ensure that a cloud server maintains the outsourced data in its original form without accessing the full data. The earliest PDP structure was suggested by Ateniese et al. [7], in which the client develops homomorphic verification tags of each block of data and outsources the tags to the cloud. In the process of auditing, a client provides the server with a challenge, which creates a piece of proof on the basis of the stored blocks and the respective tags. The customer is then able to validate the evidence to verify that he or she possesses the data.

Later research has suggested a number of enhancements to PDP schemes. Erway et al. [17] created the dynamic PDP model that facilitates data updating, such as insertion, deletion, and modification. Wang et al. [10] proposed public auditing mechanisms, which are data integrity verification by third-party auditors without access to the data. The methods enhance scalability and usability in the cloud.

Nonetheless, techniques based on PDP usually use computationally intensive cryptographic functions like modular exponentiation [18, 19] and bilinear pairings [20]. These tasks demand high processing power and memory, which might be incompatible with a lightweight device that can be used in the IoT or edge setting [10, 11]. Moreover, the cost of storage overhead due to verification metadata is proportional to the number of data blocks, posing further scalability problems.

2.3. Proof of Retrievability (PoR)

Proof of Retrievability (PoR) schemes are an extension of the concept of PDP by ensuring that the cloud server not only has the data in the cloud, but also that the complete dataset can be restored when necessary. The PoR model was initially proposed by Juels and Kaliski [8], which inserts special sentinel blocks in the data and then outsources it to the cloud. The existence of such sentinel blocks during the verification process ensures that the data are intact.

Subsequent papers by Shacham and Waters [9] suggested a more effective PoR construction based on homomorphic authenticators together with error-correcting codes. These are mechanisms that enable the clients to rebuild the original data in case some parts of the stored dataset are affected by corruption. PoR schemes are thus more effective in ensuring data availability and reliability than the PDP-based solutions.

PoR schemes also incur extra storage overhead from encoded redundancy and erasure coding, despite their benefits. Also, the encoding and decoding processes involved in error correction add complexity to the computations. These qualities render PoR systems inappropriate in settings where devices are not only less computationally endowed and have less memory capacity but also when they are integrated into IoT systems and edge computing infrastructures [21].

2.4. Lightweight Integrity Checking Methodologies

In order to overcome the shortcomings of the classic models of integrity verification, scholars have considered lightweight cryptographic protocols dedicated to resource-constrained settings. Hashing algorithms with a low weight and symmetric authentication models have been suggested to offer a lower computational level and maintain the necessary security guarantees. Hongliang Zhu et al. [14] showed that lightweight hash functions can save a lot of computation compared with conventional public-key cryptography functions.

The use of Merkle Hash Trees (MHT) has also become a common and efficient way of integrity verification. The Merkle tree structure was initially described by Merkle [16], and it is a hierarchical tree structure in which the data blocks are arranged into hierarchical branches with the internal nodes corresponding to the hash of the child nodes.

This form provides effective verification of the individual data blocks by authentication paths whose size increases logarithmically with the dataset size. As illustrated by Neenu Garg [15], the use of Merkle tree verification can greatly minimise communication overhead in cloud storage systems.

Moreover, it has been suggested to use batch auditing systems to check various data blocks at a time or for various users. Tian et al. [13] proposed a batch audit method which consolidates the verification process to enhance performance in large-scale cloud systems.

Despite the increased scalability of these methods, most available lightweight solutions are primarily aimed at minimising the computational complexity, and most do not perform a thorough analysis of security against sophisticated adversarial models.

2.5. Edge-Assisted Integrity Checking

The other potential research avenue is the utilization of the edge computing infrastructure to support integrity verification. Intermediate edge gateways in a cloud-edge architecture usually have more computational resources than single IoT devices. These gateways have the potential to do partial aggregation of verification information and send it back to centralised cloud servers.

Satyanarayanan [4] emphasised the future of edge computing in enhancing the security and efficacy of distributed systems, as it takes the burden of complex tasks off the overloaded devices. The computationally expensive functions of aggregation and proof generation can be done at the gateway node instead of the device level using edge-assisted verification models.

This hierarchical design leads to a major decrease in the default computation overhead and its communication expenses for the resource-constrained devices. Nevertheless, the models that have been developed so far do not have built-in frameworks that merge the hierarchical aggregation with lightweight cryptographic verification.

2.6. Recent Progress in Edge-Cloud Integrity Checking (2022-2025)

Recent works have been done on the design of scalable integrity checking mechanisms that are specific to distributed edge cloud. Ding et al. [22] proposed an edge data integrity verification scheme that supports dynamic data updates, and Qiao et al. [23] designed a lightweight integrity auditing method for edge data in distributed storage.

Bin Liu et al. [24] suggested a blockchain-based auditing system where data integrity is validated in IoT cloud storage systems with distributed registers. Their implementation increases the levels of transparency and resistance to tampering because verification records are kept in a blockchain network [25].

Zhang et al. [26] proposed a lightweight authentication and verification protocol for edge computing infrastructures that consumes less energy through the application of effective cryptographic primitives. On the same note, Wang et al. [27] proposed an integrity verification scheme with edge support for integrity checking in IoT clouds where some integrity checking tasks are performed at the edge.

Alruwaili [28] proposed blockchain-assisted data edge verification for an IoT-cloud platform; Zhao et al. [29] studied data integrity verification in mobile edge computing; and Ye and Jiang [30] provided data integrity verification for an edge computing environment.

Whether it is the integrity inspection in offloading in decentralised edge computing, which Seyedi et al. [31] studied, or whether it's the general survey of edge data integrity verification, as in Zhao et al. [32], upholding integrity is a key goal in other aspects of edge computing.

Recent developments further indicate an increasing concern about scalable integrity verification systems that can support modern distributed infrastructures. However, most of such solutions either are based on blockchain systems that add extra latency [33-35] or do not analyse computational complexity on a constrained device.

Consequently, this leaves a gap in the implementation of lightweight frameworks that integrate hierarchical aggregation, effective hashing functions and scalable auditing that can be used in cloud-edge ecosystems.

2.7. Research Gap

Based on the literature reviewed, there are a number of research gaps that can be defined. To start with, the majority of current integrity assurance models are based on computationally expensive public-key cryptographic services, which do not scale to resource-constrained edge devices.

Second, although lightweight hashing methods lower the computation cost, most of the models do not offer end-to-end protection against advanced attacks, including collusion and replay attacks.

Third, the reliance on PoR-based techniques presents extra overhead in storage, in the form of redundancy encoding, which might not be feasible in a large-scale system with a stream of data to be stored in the cloud.

In addition to that, a lot of current frameworks do not leverage the hierarchical structure of modern cloud-edge settings, where intermediate gateways may combine verification data, followed by communicating with cloud servers.

This still leaves the void of combined integrity verification frameworks that combine the three key factors of computational performance, communication cost, scalability, and strong adversarial properties.

The main reason why the proposed lightweight hierarchical data integrity verification framework has to address these limitations is its motivation.

A comparative summary of representative approaches is presented in Table 1.

Table 1. Comparative analysis of cloud data integrity verification approaches

Approach Type	Technique Used	Computational Overhead	Storage Overhead	Suitability for Resource-Constrained Devices	Key Limitation
Traditional Hash-Based	Full hash precomputation	Low computation, high communication	Minimal	Low	Requires full data retrieval
PDP Schemes	Homomorphic tags, modular exponentiation	High	Moderate	Limited	Heavy public-key operations
PoR Schemes	Erasure coding + authenticators	High	High	Low	Additional redundancy and complexity
TPA-Based Models	External auditing	Moderate (client), high (auditor)	Moderate	Moderate	Trust and privacy concerns
Lightweight Hash Models	Merkle trees, symmetric hashing	Low to Moderate	Low	High	Limited dynamic support
Batch Auditing	Aggregated verification	Moderate	Moderate	Moderate	Scalability-security tradeoff

2.8. Problem Statement

The usage of cloud-integrated environments over time is relying more on edge and IoT devices to create, pre-process and transfer large amounts of data to centralized cloud servers. Nevertheless, the hardware and energy constraints under which these devices run are very severe and greatly constrain their capability to undertake traditional cryptographic integrity verification schemes. An average edge sensor or embedded IoT node can have less than 256 MB of memory, and in most instances much less (e.g., 32-128 MB), and have processing units of less than 200–400 MHz. These specifications are very low compared to those of standard client machines that have been used in traditional cloud security models. It follows that the integrity verification mechanisms should be meticulously created to suit the limited memory footprint and limited computation throughput.

Besides processing constraints, there is another important constraint: communication overhead. The edge devices are often based on the use of low-rate wireless technology like LoRa, ZigBee, NB-IoT, or energy-efficient Wi-Fi protocols. Too much data transmission to support integrity auditing is likely to raise the latency, use more energy, and lower the network lifetime. Thus, the verification protocols for integrity cannot afford to waste much of the challenge response and also should not propagate large metadata structures repeatedly.

The current Provable Data Possession (PDP) schemes involve the creation and validation of homomorphic tags on the foundation of either modular exponentiation operations or pairing-based cryptographic operations. Such operations consume precious computing power and are not ideal for low-frequency processors in devices. Moreover, in the PDP model, auxiliary metadata is typically a linear function of the number of data blocks (NB), increasing memory consumption. On top of these requirements, PoR schemes add redundancy and complexity through encoding. There is a chance that these mechanisms may go out of control and result in limited edge devices running out of resources.

Lightweight hash-only integrity, on the other hand, minimises the complexity of the computations but does not support scalability or dynamic updates. Most hash-based algorithms involve either recomputing all the data when they are verified, or rely on static tree structures that are not very efficient when dealing with large-scale and continuous data streams. In addition, the standalone verification of a hash is not probabilistically efficient in auditing and might not be adequately resistant to sophisticated adversarial behaviour, including deleting selected data or colluding.

Hence, there is an obvious technical divide between heavyweight integrity checking schemes and the excessively naive hash-based techniques. To overcome the problem, a practical solution should meet low computational complexity, low memory usage, low communication overhead, and scalable verification in resource-constrained cloud-edge systems. The need to fill this gap is the key driving force of the proposed study.

The comparison in Table 1 highlights the different objectives and trade-offs of existing integrity verification approaches. Traditional hash-based methods offer simplicity but may require extensive data processing or communication when verification depends on recomputing hashes over large datasets. PDP and PoR provide established remote-storage verification capabilities, although their computational and storage requirements vary according to the construction employed. Lightweight hashing reduces the cost of individual integrity operations, whereas Merkle trees provide structured authentication paths for block-level verification. Edge-assisted processing and batch auditing complement each other by distributing computation and reducing repeated communication. However, these techniques address different aspects of the

verification problem and should not be assumed to provide equivalent security guarantees. The proposed framework builds on these established mechanisms by coordinating lightweight integrity processing, hierarchical Merkle-tree authentication, and batch-auditing operations across the edge–gateway–cloud architecture. Its contribution is evaluated through a common set of performance metrics under the specified experimental conditions. A fair comparison consequently requires consistent workload sizes, challenge configurations, security assumptions, and measurement procedures for the proposed framework and its baselines.

3. Proposed Methodology

This part introduces a lightweight hierarchical data integrity checking system that can be inherently made to work with resource-constrained edge devices that are deployed in a cloud-integrated platform. The main goal of the framework is to ensure good data integrity with low computational complexity, communication overhead, and storage load on the limited devices. In contrast to the conventional integrity mechanisms involving a lot of reliance on public-key cryptographic operations, the suggested framework allocates the responsibilities among the architectural layers and uses lightweight homomorphic hashing with hierarchical Merkle tree aggregation. This design will make integrity checking scalable, efficient, and able to meet the hardware constraints of the IoT and edge nodes. As illustrated in Figure 1, the general architecture is based on the multi-tier model, comprising edge devices, an edge gateway, cloud storage, and a verifier module. Edge devices are in charge of processing data blocks and producing them. Because their memory (usually less than 256 MB) and processors (often less than 400 MHz) are limited, they perform minimal hash computation and cannot handle computationally expensive operations like modular exponentiation or pairing-based cryptography. The calculated hash values are sent to an edge gateway, which is relatively more powerful in its calculations. The gateway aggregates the received hash values and builds a hierarchical Merkle tree over them. Both the original data blocks and their metadata integrity are stored in the cloud storage server. The verifier (who can be the owner of data or an appointed auditing authority) issues integrity challenges on the cloud periodically and verifies the delivered proofs.

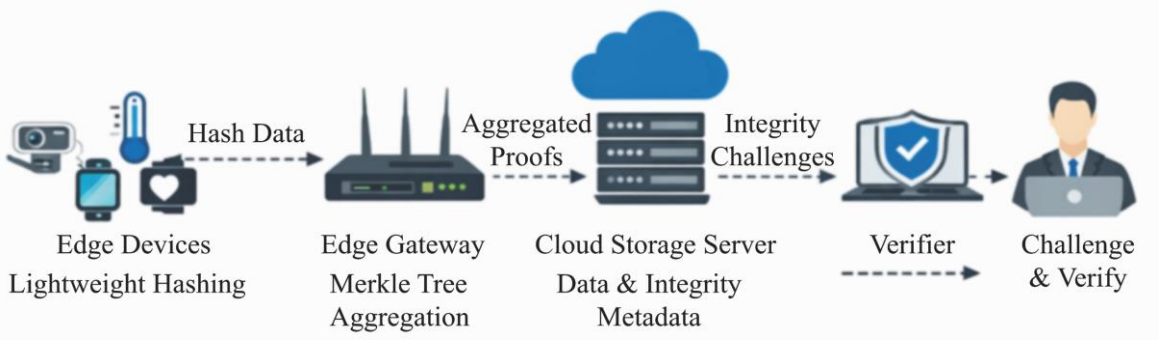

Fig. 1 Lightweight hierarchical integrity verification architecture

3.1. System and Threat Model

There are 4 roles in the framework. Edge devices: produce data blocks and compute one hash value for each block. The edge gateway is a trustworthy aggregation point in the data owner's domain, to which the block hash values are sent via an authenticated connection; the data owner constructs the Merkle tree and reveals its root to the gateway. The cloud storage server holds the blocks and the tree and responds to audit challenges, but is not trusted. The challenges are issued by the verifier to whom only the authenticated root is known, either the data owner himself or a delegation. Every edge device identifier also validates an aggregate from its blocks and submits it directly to the verifier.

An attacker – The adversary is the cloud server, or an attacker who has compromised the cloud server. It can corrupt, replace, or delete blocks and tree nodes, and can return stale data from an earlier version of the tree, and can make adaptive choices regarding its responses; however, it operates in probabilistic polynomial time. The following are three assumptions. Each edge and the verifier share a key: the edge-to-gateway link and the gateway-to-verifier channel authenticate the block hash values and the root (e.g. using a lightweight MAC). After sampling, Merkle audits do not require the gateway to be controlled by the adversary, or to be colluding with the cloud server, which is the case for range audits (Section 3.5). At challenge time, proofs are generated from the blocks stored on the server; the server has no separate cache of the leaf values in the blocks that are no longer stored. Assumption A3 and its consequences are discussed in Remark 1. In this work, confidentiality is out of scope; if we encrypt the blocks prior to outsourcing, then we verify integrity when served in the encrypted format.

3.2. Notation and Primitives

Let $D = \{d_1, \ldots, d_n\}$ be the outsourced data blocks produced by the edge devices, and let v_i be the version counter of block d_i ($v_i = 0$ for the initial upload). Let $H_0 : \{0,1\}^* \rightarrow \{0,1\}^{\ell}$ be a collision-resistant hash function, instantiated with SHA-256 so that $\ell = 256$, and let p be a prime with $p > 2^{\ell}$. The map $\langle \cdot \rangle$ interprets an ℓ-bit string as an integer in $\{0, \ldots, 2^{\ell} - 1\} \subset \mathbb{Z}_p$; it is injective because $2^{\ell} < p$, so working modulo p loses no information. The symbol $\|$ denotes concatenation, with the version and the index encoded as fixed-length fields so that the encoding of (v_i, i, d_i) is unambiguous. A challenge is a set $C \subseteq \{1, \ldots, n\}$ of k distinct indices; $L = \lceil \log_2 n \rceil$ is the height of the Merkle

tree, and ρ is the fraction of blocks that the server has corrupted or deleted. The hash function H_0 is collision resistant if no probabilistic polynomial-time adversary can find $x \neq x'$ with $H_0(x) = H_0(x')$ except with negligible probability; this probability is written $\mathrm{Adv}_{\mathrm{coll}}$.

The Message Authentication Code (MAC) is existentially unforgeable under chosen-message attacks, with forger advantage $\mathrm{Adv}_{\mathrm{MAC}}$, and $\mathrm{Adv}_{\text{R-SIS}}$ denotes the advantage in solving the ring short-integer solution problem defined in Section 3.3.

3.3. Lightweight Additive Homomorphic Hash

Definition 1 (Block hash). The block hash of block d_i with version v_i is

$$h_i = \langle H_0(v_i \| i \| d_i) \rangle \in \mathbb{Z}_p \tag{1}$$

Definition 2 (Aggregate hash). For a challenge set C, the aggregate hash of the blocks indexed by C is

$$\sigma_C = \sum_{i \in C} h_i \mod p \tag{2}$$

The aggregate has a fixed size of $\lceil \log_2 p \rceil$ bits regardless of k, and it is additively homomorphic: for disjoint sets C_1 and C_2,

$$\sigma_{C_1 \cup C_2} = \sigma_{C_1} + \sigma_{C_2} \pmod p \tag{3}$$

Consequently, a gateway can merge partial aggregates from several edge devices by modular addition without recomputing any block hash, and when one block changes from h_i to h'_i, a stored aggregate that covers it is updated in constant time.

Only one call to H_0 per block and modular additions are needed; unlike the RSA-based authenticators [19] and bilinear pairings [20] used in many PDP and PoR constructions, no exponentiation or pairing is involved. The construction follows the additive set-hashing idea of Bellare et al. [36].

The modular sum is homomorphic but not binding. An adversary can replace two leaf values h_a and h_b by $h_a + \delta$ and $h_b - \delta$ and leave σ_C unchanged; this algebraic cancellation is also the idea behind Wagner's generalised birthday attack on additive hashes [37]. For this reason, the soundness of the sampled audit never relies on σ_C (Theorems 1 and 2): the Merkle commitment binds every returned value, and the modular sum only serves as a compact consistency check.

To obtain an aggregate that is binding by itself, the framework adds an algebraic construction. A Pedersen commitment [38] is additively homomorphic and binding under the discrete logarithm assumption, but it needs modular exponentiation in a large group, which is the cost this work removes from the edge. Therefore, use a position-keyed aggregate based on the ring Short-Integer-Solution (SIS) problem [39, 40], in the style of the SWIFFT hash [41].

Definition 3 (Hardened aggregate). Let $R_q = \mathbb{Z}_q[X]/(X^N + 1)$ with $N = 256$ and the prime $q = 7681$, so that $q \equiv 1 \pmod{2N}$ and products are computed with the number-theoretic transform.

For every position i, let $a_i \in R_q$ be a public ring element obtained by expanding a public seed together with i with the hash function. The 256 bits of the block hash h_i are read as a polynomial η_i with coefficients in $\{0, 1\}$. The hardened image of block i and the hardened aggregate over a set C are

$$g_i = a_i \cdot \eta_i \in R_q \tag{4}$$

$$\sigma^*_C = \sum_{i \in C} g_i \in R_q \tag{5}$$

The hardened aggregate is additively homomorphic in R_q, has a constant size of $N\lceil \log_2 q \rceil = 3{,}328$ bits (416 bytes), and costs one ring product per block. Because every position has its own key a_i, moving value from one position to another no longer cancels.

The output length is chosen so that the generalized birthday estimate of [37] for position-keyed sums over ranges of up to 10^6 blocks exceeds 2^{128}; this is a heuristic estimate for one known attack, and the parameters should be checked against lattice-reduction estimators before deployment.

3.4. Hierarchical Merkle Aggregation

The framework uses hierarchical aggregation of Merkle trees at the edge gateway level in order to scale up the verification process. As illustrated in Figure 2, the gateway arranges the block hash values as the leaf nodes of a Merkle Hash Tree (MHT) [16].

Formally, the leaf level is padded with a fixed dummy value to 2^L leaves. The leaf nodes and the internal nodes are defined as

$$u_{0,i} = H_0(0\text{x}00 \| i \| h_i) \tag{6}$$

$$u_{j,t} = H_0(0\text{x}01 \| u_{j-1,2t-1} \| u_{j-1,2t}) \tag{7}$$

for $1 \leq j \leq L$ and $1 \leq t \leq 2^{L-j}$, and the root is $R = u_{L,1}$. The prefixes 0x00 and 0x01 separate leaf inputs from internal-node inputs, so a leaf can never be confused with an internal node. The gateway sends R to the verifier together with a counter ctr that increases with every committed update and a message authentication code over ($R \Vert$ ctr); the verifier keeps only the root with the highest authenticated counter.

The root acts as a compact commitment to the whole dataset. Any change to a data block modifies the corresponding leaf hash value, and the change propagates upwards to the root. Integrity checking is therefore carried out by recomputing the authentication path between a challenged leaf and the root and comparing the result with the stored commitment.

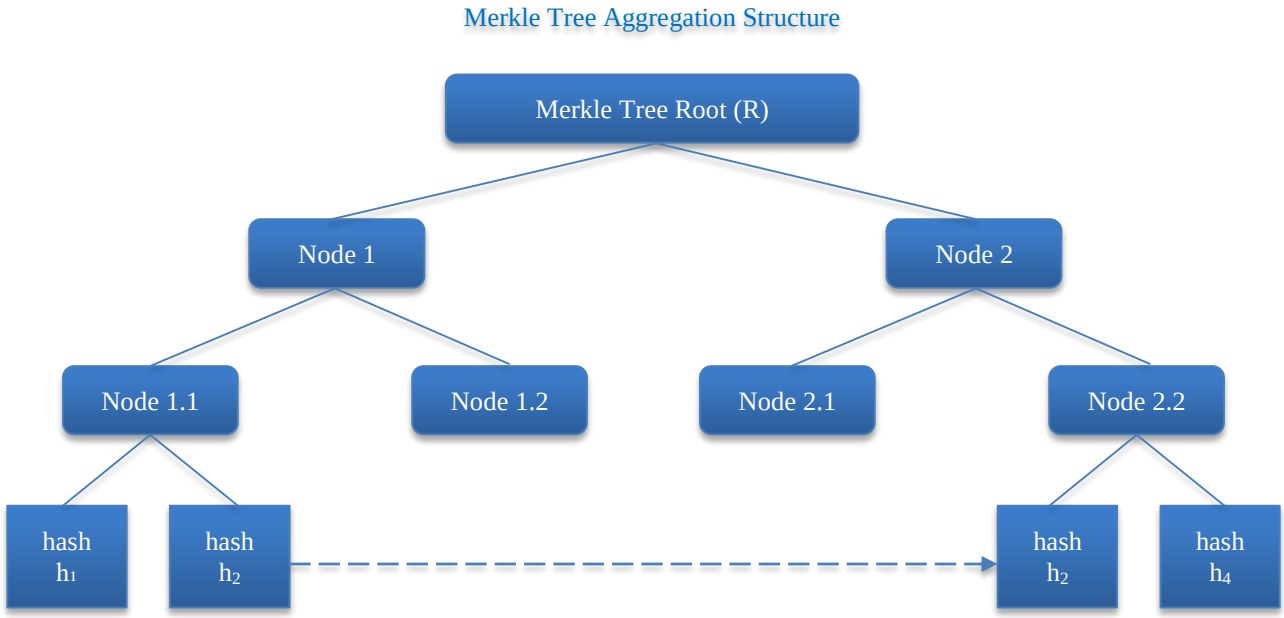

Fig. 2 Hierarchical merkle tree aggregation structure

3.5. Audit Protocol

The protocol has 4 phases. Setup (Algorithm 1): Each edge device calculates the block hash of blocks and sends it to the gateway to construct a Merkle tree and return it to the verifier; blocks and the tree are stored on the cloud server. The verifier starts the audit process: In the Challenge, the verifier chooses k random and different blocks' indexes, a set C in the set B.

In the Prove protocol (Algorithm 2), the cloud server re-generates the block hash of each challenged block from its stored information, concatenates all these hashes to form the proof value σ and returns σ and the challenged leaf values together with a batch authentication path, which consists of the sibling nodes used to reconstruct the root for all leaves that were challenged; siblings that appear multiple times in a particular batch authentication path are only returned once. During Verify (Algorithm 3), the verifier verifies σ with the provided leaf values and compares the reconstructed root with the authentic root. To avoid repetitive communications between the verifier and the cloud server, the batch mechanism validates multiple blocks together, and decreases bandwidth consumption. Besides sampled auditing, one option is range auditing (Algorithm 4), which will enable the verifier to check all blocks of one edge device with a constant-size proof; in particular, the proof is sent to the verifier directly by each edge device by hashing over the blocks, with the help of MAC.

The four procedures are listed as Algorithms 1-4.

Algorithm 1. Setup and commitment (Edge device, gateway, and verifier)

Input: blocks $d_1, \ldots, d_n$ (block i belongs to device j if $i \in B_j$), versions v_i, hash function H_0, prime p, keys a_i

Output: block hashes, Merkle root R, device aggregates G_j, authenticated commitments

Edge device j

1: **for** each block d_i with $i \in B_j$ **do**
2: $h_i \leftarrow \langle H_0(v_i \| i \| d_i) \rangle$; send (i, h_i) to the gateway
3: $g_i \leftarrow a_i \cdot \eta_i$ in R_q
4: **end for**
5: $G_j \leftarrow \Sigma_{i \in Bj}\, g_i$; send $(j, G_j, \text{ctr}_j, \text{MAC}_{Kj}(j \| \text{ctr}_j \| G_j))$ to the verifier

Gateway

6: $u_{0,i} \leftarrow H_0(0\text{x}00 \| i \| h_i)$ for $i = 1, \ldots, n$; pad the leaf level to 2^L leaves
7: **for** $j = 1$ **to** L and $t = 1$ **to** 2^{L-j} **do**
8: $u_{j,t} \leftarrow H_0(0\text{x}01 \| u_{j-1,2t-1} \| u_{j-1,2t})$
9: **end for**
10: $R \leftarrow u_{L,1}$; send $(R, \text{ctr}, \text{MAC}(R \| \text{ctr}))$ to the verifier
11: upload the blocks and all tree nodes to the cloud server; **return** R

Algorithm 2. Proof generation (Cloud server)

Input: challenge set C with $|C| = k$, stored blocks and versions, stored tree
Output: proof $(\sigma, \{h_i\}_{i \in C}, \Psi)$

1: $\sigma \leftarrow 0$
2: for each index i in C do
3: $h_i \leftarrow \langle H_0(v_i \| i \| d_i) \rangle$, computed from the stored block
4: $\sigma \leftarrow (\sigma + h_i) \bmod p$
5: end for
6: $\Psi \leftarrow$ sibling nodes on the paths from the leaves in C to the root, each node included once
7: return $(\sigma, \{h_i\}_{i \in C}, \Psi)$

Algorithm 3. Batch verification (Verifier)

Input: challenge set C, proof $(\sigma, \{h_i\}_{i \in C}, \Psi)$, authenticated root R with the latest counter
Output: Accept or Reject

1: if $\sigma \neq \Sigma_{i \in C} h_i \bmod p$ then return Reject
2: $u'_{0,i} \leftarrow H_0(0x00 \| i \| h_i)$ for each i in C
3: recompute the ancestors level by level up to level L, taking the left/right order of each pair from the binary expansion of $i - 1$ and the missing siblings from Ψ
4: if a required sibling is missing or Ψ contains a node that is not used, then return Reject
5: if the recomputed root $R' = R$ then return Accept else return Reject

Algorithm 4. Device-Level range audit (Cloud server and verifier)

Input: device index j with block range B_j; the verifier holds (j, G_j, ctr_j, MAC tag) with the latest counter
Output: Accept or Reject
Cloud server

1: $\Gamma \leftarrow 0$
2: for each i in B_j do $h_i \leftarrow \langle H_0(v_i \| i \| d_i) \rangle$ from the stored block; $\Gamma \leftarrow \Gamma + a_i \cdot \eta_i$ end for
3: return Γ

Verifier

4: verify the MAC of (j, ctr_j, G_j) under K_j and check that ctr_j is the latest counter; otherwise return Reject
5: if $\Gamma = G_j$ then return Accept else return Reject

3.6. Security Analysis

The analysis uses the system model and the assumptions A1–A3 of Section 3.1. Theorems 1–4 concern the sampled Merkle audit, and Theorems 5 and 6 concern the hardened aggregate and the range audit.

3.6.1. Theorem 1 (Leaf Binding)

Let $(v, i, d) \neq (v', i', d')$ be two inputs to the block hash. If both inputs give the same block hash value, then a collision in H_0 has been found.

Proof. Because $2^{\ell} < p$, the map $\langle \cdot \rangle$ is injective, so equal block hash values imply $H_0(v \| i \| d) = H_0(v' \| i' \| d')$. The encoding is unambiguous, so the two arguments are different strings, and together they form a collision for H_0.

3.6.2. Theorem 2 (Soundness of Batch Verification)

Let R be the authentic root of a tree built over the leaf values $h_1, \ldots, h_n$. Suppose a probabilistic polynomial-time adversary, which may have modified any stored block or tree node, outputs a challenge set C and a proof that Algorithm 3 accepts against R, in which at least one submitted value, that is a returned leaf value or a sibling node in Ψ, differs from the true one. Then the adversary yields a collision in H_0, so the acceptance probability is at most Adv_{coll}, which is negligible.

Proof: The verifier fixes the tree height at L, derives the left/right position of every node from the binary expansion of the index, and rejects a proof with missing or unused nodes, so the position of every submitted value is fixed. Call a recomputed node bad if it differs from the true node at the same position. A returned leaf value that differs from the true one has a leaf input $0x00 \| i \| h$ that differs from the true input; if the two hashes are equal, this is a collision, and otherwise the leaf node is bad. A sibling in Ψ that differs from the true node gives its parent an input that differs from the true input. In both cases, some node is computed from an input that differs from the true input. If its output equals the true output, this is a collision in H_0; otherwise the node is bad, and the same argument applies to its parent. Because the proof is accepted, the recomputed root equals the true root, so at some level a node computed from different inputs has the true output, which is a collision in H_0. The prefixes 0x00 and 0x01 prevent a leaf input from being reinterpreted as an internal-node input.

3.6.3. Theorem 3 (Detection Probability)

Suppose that the server has corrupted or deleted $x = \lceil \rho n \rceil$ blocks and that the verifier challenges k distinct indices chosen uniformly at random. The probability that the challenge contains at least one affected block is

$$P_{\text{det}} = 1 - \frac{\binom{n-x}{k}}{\binom{n}{k}} \tag{8}$$

and it satisfies

$$P_{\text{det}} \geq 1 - (1-\rho)^k \tag{9}$$

Proof: The probability that all k challenged indices avoid the x affected blocks is the binomial ratio in Equation (8), which equals $\prod_{m=0}^{k-1} (n-x-m)/(n-m)$. Each factor is at most $(n-x)/n \leq 1-\rho$, which gives Equation (9). Under A3 and Theorem 2, every challenged block that was corrupted or deleted leads to rejection except with probability at most Adv_{coll}: a corrupted block produces a leaf value that differs from the committed one, and a deleted block cannot be hashed at all. The audit therefore rejects with probability at least $P_{\text{det}} - \text{Adv}_{\text{coll}}$. For example, challenging $k = 460$ blocks detects the corruption of 1% of the blocks ($\rho = 0.01$) with probability at least 0.99.

3.6.4. Theorem 4 (Replay and Rollback Resistance)

Let the verifier hold the root, or the device aggregate, with the highest authenticated counter it has received. If block i was updated after version v, a proof generated from the stale state (v, d^{old}) of that block is rejected except with probability at most Adv_{coll}. A root or aggregate authenticated with a lower counter is rejected, and the adversary can make the verifier accept a counter higher than the true one only by forging the MAC, which succeeds with probability at most Adv_{MAC}.

Proof: The stale leaf value is $\langle H_0(v \parallel i \parallel d^{\text{old}}) \rangle$ and the current one is $\langle H_0(v_i \parallel i \parallel d_i) \rangle$ with $v < v_i$. The two values differ unless H_0 has a collision (Theorem 1), and Theorem 2 then gives rejection. A root or aggregate with a lower counter is not accepted, because the verifier keeps only the one with the highest authenticated counter, and a higher counter requires a valid MAC. A recorded proof for the current state can be replayed without harm, since it vouches for unchanged data; it answers a fresh random challenge only if the new challenge set equals the recorded one, which has probability $1/\text{C}(n, k)$.

3.6.5. Theorem 5 (Binding of the Hardened Aggregate)

Let the ring elements a_i be uniformly random. Suppose a probabilistic polynomial-time adversary, given them, outputs a set C and two different assignments of digest polynomials $(\eta_i)_{i \in C} \neq (\eta'_i)_{i \in C}$ with the same hardened aggregate. Then it solves the ring-SIS problem for the keys $(a_i)_{i \in C}$ with infinity-norm bound 1, so its success probability is at most $\text{Adv}_{\text{R-SIS}}$. In particular, the cancellation of Section 3.3 is not possible without solving that problem.

Proof: Let $\Delta_i = \eta_i - \eta'_i$. Its coefficients lie in $\{-1, 0, 1\}$, and not all Δ_i are zero. Equal aggregates give $\Sigma_{i \in C}\, a_i \cdot \Delta_i = 0$ in R_q, which is a non-zero short solution of the ring-SIS instance defined by the keys. The keys are obtained by hash expansion of the public seed and the position, which is modelled as a random oracle.

3.6.6. Theorem 6 (State Binding under Gateway Collusion)

Let an adversary control the cloud server and the gateway, but not edge device j or its key K_j. Let S be the blocks of device j and $S' \neq S$ the blocks that the adversary stores for the range B_j. If the range proof Γ is computed from S' as in Algorithm 4 and the verifier accepts, then the adversary has forged the MAC, found a collision in H_0, or solved ring-SIS. The success probability is at most $\text{Adv}_{\text{MAC}} + \text{Adv}_{\text{coll}} + \text{Adv}_{\text{R-SIS}}$.

Proof: By the unforgeability of the MAC, the pair (G_j, ctr_j) held by the verifier was produced by device j, which computed G_j from the digests of S. Acceptance means $\Gamma = G_j$. Because $S' \neq S$, some block or version differs, and by Theorem 1 its digest differs unless H_0 has a collision. The two digest assignments are then different and have the same hardened aggregate, so Theorem 5 yields a ring-SIS solution. A union bound over the three events gives the stated probability.

Remark 1 (Scope of the Guarantee)

Theorems 1–6 show that an accepted proof is consistent with the committed values, that corruption, tampering with stored tree nodes, replay, and rollback are detected with the stated probability, and that a colluding gateway and cloud server cannot present modified blocks as the device's data. They do not establish possession or retrievability in the sense of PDP and PoR: a server that keeps the leaf values and discards the blocks, which violates A3, would still produce accepted proofs, and because the gateway holds all leaf values, a colluding gateway and cloud server can answer without storing the blocks. Theorem 6 therefore binds what the adversary presents as stored state, not whether it holds the data. Closing this gap requires keyed homomorphic tags that only the data owner can verify, as in the private-verification PoR scheme of Shacham and Waters [9], at the cost of larger proofs; this is left for future work (Section 6.3).

3.7. Complexity Analysis

Let b denote the block size in bytes. Each edge device evaluates H_0 once per block, which takes time linear in b and requires no modular exponentiation. Building the tree at the gateway takes $O(n)$ hash evaluations once, and the update of a single block re-hashes one leaf-to-root path, $O(\log n)$. In the Prove phase, the cloud server computes k block hashes and $k-1$ modular additions and collects at most kL sibling nodes, which costs $O(kb + k \log n)$. In the Verify phase, the verifier performs $k-1$ modular additions and at most kL node hashes, that is $O(k \log n)$ hash evaluations; ancestors shared by several challenged leaves are hashed only once, so the actual cost is lower when the challenged leaves are close together.

The proof consists of one element of $\mathbb{Z}_p$ (the aggregate σ), k leaf values, and at most kL sibling nodes, so its size is $O(k \log n)$ hash values. For a fixed audit size k, it grows only logarithmically with the number of blocks and does not depend on the block size b, whereas returning the challenged blocks themselves would cost $O(kb)$. By comparison, proof computation and verification in PDP and PoR schemes involve a number of modular exponentiations or pairing operations that grows with k [7, 9]. The cloud stores the blocks, the leaf values, and the internal nodes, so the metadata storage is $\Theta(n)$; no erasure coding is used, in contrast to PoR schemes.

Sultana and Srinivas [43] proposed a visual cryptography-based approach for protecting data privacy in cloud computing. The methodology integrates image fusion, data embedding, LSTM, ZigZag scanning, and Discrete Cosine Transform (DCT) to enhance image security. The proposed framework achieved promising results, with a PSNR of 55.9 dB and SSIM of 0.98, demonstrating its effectiveness in protecting confidential visual information. The hardened aggregate adds one product in R_q per block, which the number-theoretic transform computes with $O(N \log N)$ modular operations ($N = 256$) after the expansion of the key a_i; this is small compared with hashing a block of several kilobytes. Updating a device aggregate after a block change takes one subtraction and one addition in R_q. A range audit costs the cloud server $O(|B_j|(b + N \log N))$, but its proof is a single ring element of 416 bytes and the verifier performs one MAC check and one comparison, $O(1)$. The prototype evaluated in Section 5 implements the sampled Merkle audit only; the hardened aggregate and the range audit are analysed here but were not part of the measured experiments. In summary, the framework removes public-key operations from the edge devices, delegates aggregation to the gateway, and combines batch auditing with Merkle authentication. Its guarantees are those of Theorems 1–6 under assumptions A1–A3, and its cost grows with the audit size k and logarithmically with the dataset size n.

4. Experimental Setup

This section gives the experimental setup followed in order to find the performance of the proposed Lightweight hierarchical data integrity verification framework. The testing is premised on the computational performance, communication cost, storage usage and scaling in a realistic cloud-edge community. The experiments were designed to replicate the environment of an interaction of resource-constrained edge devices, an edge gateway and a cloud storage server in a distributed environment.

4.1. System Architecture and Hardware Configuration

The experimentation was configured in a manner that imitated a cloud–edge architecture in the real world. The layer of edge devices was designed as a model that depicts strained IoT hardware capabilities. Every edge node had 128 MB RAM, a 300 MHz processor of equivalent speed, and low transmission bandwidth (1–5 Mbps). These requirements are in line with the common microcontroller-based IoT systems deployed in smart monitoring and sensing technologies.

The edge gateway was made with increased computing power, having 8 GB RAM and a 2.6 GHz quad-core processor. Infrastructure-as-a-Service (IaaS) cloud conditions were simulated by the cloud storage server in a virtualised environment with 16 GB RAM and an expandable storage capacity. The verifier module was implemented as a simple client that can provide random integrity challenges and authenticate the returned proofs.

The hardware configuration of all four components is summarised in Table 2.

Table 2. Hardware configuration of experimental environment

Component	Processor Configuration	Memory Capacity	Network Bandwidth	Role in Framework
Edge Device	300 MHz (simulated MCU)	128 MB	1–5 Mbps	Data generation & lightweight hashing
Edge Gateway	2.6 GHz Quad-Core CPU	8 GB	100 Mbps	Merkle aggregation & proof coordination
Cloud Server	Virtualized 4 vCPU instance	16 GB	1 Gbps	Data storage & proof generation
Verifier	2.4 GHz CPU	8 GB	100 Mbps	Challenge issuance & verification

4.2. Generation and Reproducibility of Data

In order to achieve reproducibility and realistic analysis, synthetic datasets were utilised along with publicly available benchmark datasets in the experimental analysis.

Synthetic datasets were created in an attempt to mimic the common workloads caused by IoT sensing devices working in edge conditions. The datasets that were generated consisted of structured data of the environmental monitoring parameters, including temperature readings, high humidity indicators, device identifiers, timestamps, and telemetry logs. These characteristics were chosen in order to depict typical data patterns produced by IoT-based monitoring systems.

To ensure consistency in the experimental results, the dataset was segmented into fixed block sizes, and integrity verification operations were performed. Block sizes between 4 KB and 64 KB were used, with an average of 16 KB in the reported runs, to cover the range of packet sizes

that are often sent out by edge devices. Smaller blocks are used to model specific sensor transmissions, and larger blocks are used to model the aggregated telemetry information produced by gateway nodes.

The total volume of data was gradually increased from about 16 MB (1,000 blocks) to about 1.6 GB (100,000 blocks) at an average block size of 16 KB to assess the effectiveness of the proposed framework with the increase in the volume of outsourced data. This setup enabled scalability to be analysed with different workloads.

Besides synthetic datasets, the ToN-IoT Telemetry Dataset was also used as an experimental dataset, as it is an open-source resource intended to be used to test the security of IoT and edge computing systems. The dataset includes heterogeneous telemetry data from distributed IoT devices together with network and system logs [42]. This data was used to ensure that the experimental assessment was captured by actual data distributions and operational patterns in the real world.

All the data were divided into fixed-size blocks before computing hash values and building Merkle trees. The total number of blocks, *n*, ranged from 1,000 to 100,000, which allowed the growth of the verification cost to be studied as a function of dataset size.

4.3. Implementation Environment

The suggested framework was developed with Python 3.10, which offers a wide range of support for cryptographic modelling and large-scale processing of data. Lightweight hash modules and Merkle tree aggregation algorithms were also designed based on optimised hashing libraries in order to replicate the behaviour of low-power devices.

In order to model communication constraints, namely latency and bandwidth, to mimic realistic network conditions in a cloud-edge environment, the NS-3 network simulator was used. This enabled the experiments to simulate realistic network delays and bandwidth constraints that are typically witnessed in the distributed IoT infrastructure.

Also, some validation experiments were run in MATLAB to triangulate measurements of computational overhead and to verify that the simulation results are accurate. This validation was done on two platforms to guarantee that the results of the experiment were consistent and reproducible.

4.4. Metrics of Performance Evaluation

Four major performance metrics were used to assess the performance of the proposed framework:

4.4.1. Computational Cost

Computational cost is the time required to perform integrity-check operations. This involves the time needed to compute hash values of data blocks, building Merkle trees, computing aggregated proofs and integrity responses.

4.4.2. Communication Overhead

The total sum of data between system components in the subsystems of challenge-response verification is known as communication overhead. It was quantified by counting the number of bytes sent between the cloud server and verifier and between the edge gateway and the cloud server in auditing operations.

4.4.3. Storage Overhead

Storage overhead is the extra metadata to support integrity verification mechanisms. This comprises storage of hash values, authentication paths, and Merkle tree nodes other than the actual blocks of data.

4.4.4. Consumption of Energy Estimation

An operation-count model was used to estimate energy efficiency with reference to standard values of energy consumption of cryptographic operations on small processors. This was approximated by the number of hashing, modular computations and communication interactions in the process of verifying each.

These assessment measures enable gaining a full picture of the performance features of the suggested framework and can compare it to current techniques of integrity verification applied to cloud storage systems.

All timings and communication overhead are given in milliseconds, while all storage overhead is given as a percentage of the raw data size. The energy estimates (energy values) are in millijoules (mJ) per audit; they are estimates based on the operation-count model in Section 5.5, not measured values from real hardware. All reported values are the average of ten individual runs, and the reduction in the amount provided by the proposed framework relative to the baseline is calculated as (baseline minus proposed value) / baseline and rounded to the nearest per cent.

5. Performance Evaluation

5.1. Experimental Setting

This section compares the performance of the proposed Lightweight Hierarchical Data Integrity Verification Framework with classical Provable Data Possession (PDP) and Proof of Retrievability (PoR) schemes. The evaluation focuses on computational cost, communication overhead, storage overhead, and energy consumption using realistic IoT data workloads. The experiments used the ToN-IoT Telemetry Dataset, a publicly available dataset of sensor and system telemetry data suitable for cloud-edge workload simulation:

https://research.unsw.edu.au/projects/toniot-datasets. All three schemes were implemented and evaluated under the hardware and protocol configuration described in Section 4, and every reported value is the mean of ten independent runs (Section 5.7).

5.2. Computational Cost

The first performance attribute is "computational cost. Proof-generation time was measured, as was verification time, for the datasets of 10,000 blocks, 50,000 blocks, and 100,000 blocks. The proposed framework required an average of 94ms when generating a proof on the cloud server, versus 182ms for the PDP baseline and 215ms for the PoR baseline, for 10,000 blocks of an average 16 KB block size, reducing time by approximately 48% and 56% compared to the PDP baseline and PoR baseline, respectively. The proof-generation times of the PDP and PoR increased to 1,480 ms and 1,732 ms per 100,000 blocks, respectively, whilst the proposed framework required 712 ms (which was a reduction of nearly 52% and 59%). There was a similar trend in the verification times: for three dataset sizes, the proposed framework was compared with PDP and PoR, and the results showed that the proposed framework was faster than PDP and PoR by 46-49% and 54-58%, respectively, due to Merkle-path verification and lightweight hash aggregation, while PDP and PoR incurred almost linear verification costs with the challenge size (Table 3).

As Figure 3 shows, the proposed framework consistently has lower computational overhead across all dataset sizes than conventional integrity models.

Table 3. Computational cost comparison across integrity models

Dataset Size (Blocks)	PDP Proof Gen (ms)	PoR Proof Gen (ms)	Proposed Proof Gen (ms)	PDP Verify (ms)	PoR Verify (ms)	Proposed Verify (ms)
10,000	182	215	94	76	89	41
50,000	845	981	404	312	375	158
100,000	1,480	1,732	712	584	698	300

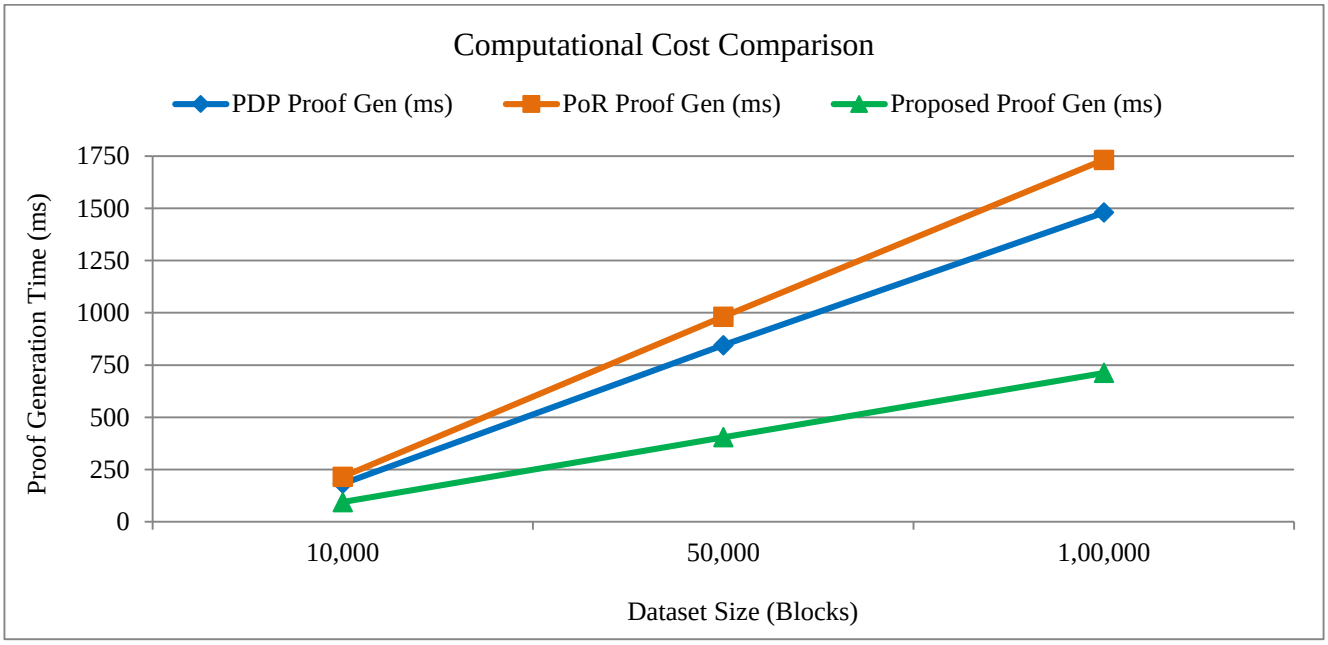

Fig. 3 Computational cost vs dataset size

5.3. Communication Overhead

Communication overhead was defined as the number of bytes communicated in a single challenge–response audit cycle. The traditional PDP or PoR involve multiple transmissions of tags, responses or encoded redundancy, resulting in a higher communication cost. PDP and PoR sent about 6.8 KB and 8.1 KB of verification data, respectively, for 10,000 blocks. The proposed scheme required approximately 3.4 KB, which is about 50% and 58% less than PDP and PoR, respectively. Since the size of authentication information grows logarithmically with the number of blocks (Table 4), the PDP and PoR overhead increased slightly to 9.5KB and 11.2KB, respectively, while the size of one authentication information in the proposed scheme was 4.7KB (51% and 58% reduction were achieved in comparison with the PDP and PoR schemes, respectively).

As Figure 4 shows, the proposed framework consistently incurs significantly lower communication overhead across all dataset sizes than conventional integrity verification schemes.

Table 4. Communication overhead comparison

Dataset Size (Blocks)	PDP (KB)	PoR (KB)	Proposed (KB)	Reduction vs PDP	Reduction vs PoR
10,000	6.8	8.1	3.4	50%	58%
50,000	8.7	10.3	4.1	53%	60%
100,000	9.5	11.2	4.7	51%	58%

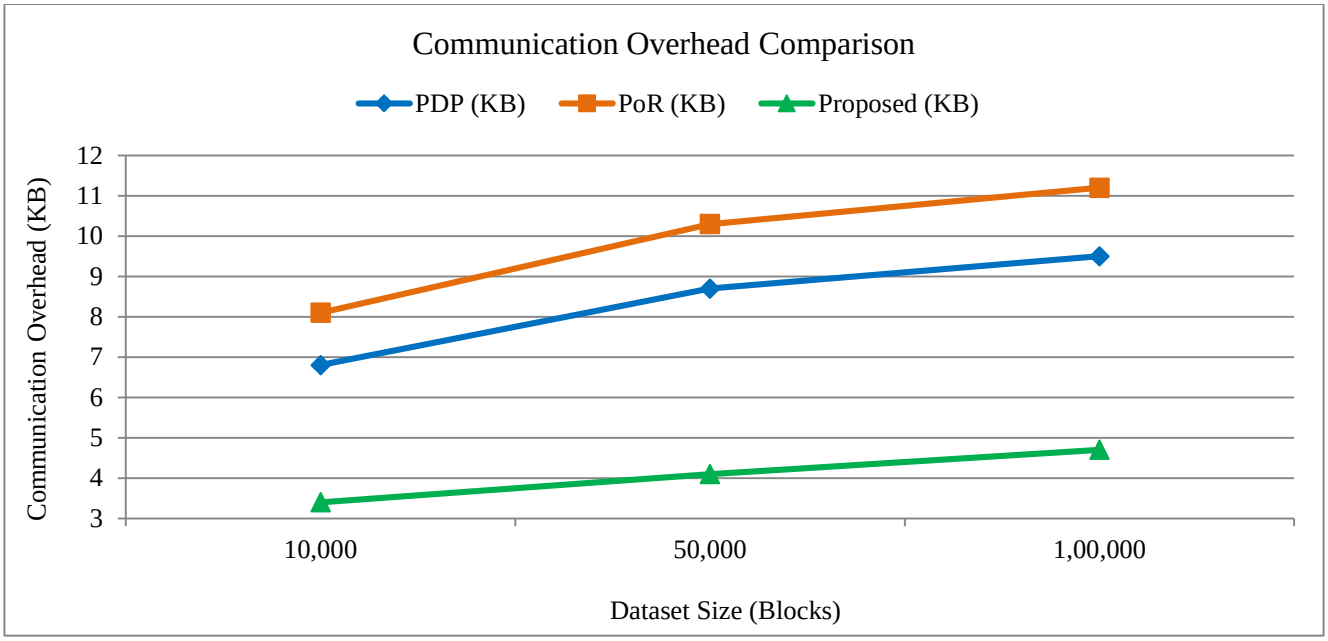

Fig. 4 Communication overhead vs dataset size

5.4. Storage Overhead

Implementing features like metadata or embedding the data offloading instruction in the file caused metadata overhead, and this was determined as a percentage of the raw data size. PDP stores an additional homomorphic tag with each block, requiring an additional storage capacity of approximately 10.4%. On top of the tagging, PoR provides erasure-coded redundancy, increasing the overhead to around 18.6%. This proposed framework only stores the block hash values and the Merkle tree nodes, which is about 40% less than PDP and 67% less than PoR (Table 5), which is very suitable for large-scale storage systems.

As Figure 5 shows, the proposed framework has the lowest storage overhead among the compared integrity schemes, making it suitable for resource-constrained cloud-edge environments.

Table 5. Storage overhead comparison

Scheme	Metadata Type	Storage Overhead (%)
PDP	Homomorphic tags	10.4
PoR	Tags + Erasure coding	18.6
Proposed	Hash values + Merkle nodes	6.2

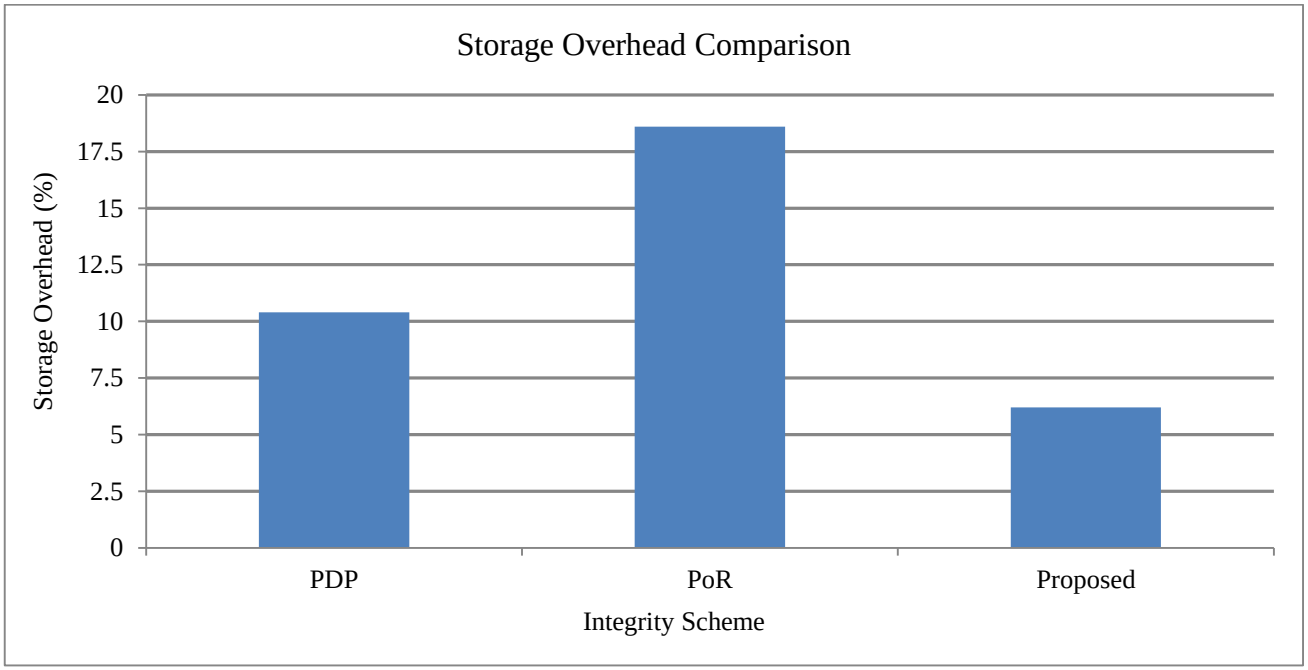

Fig. 5 Storage overhead distribution

5.5. Energy Consumption

To approximate energy consumption, we used operation-level modelling based on the known energy cost per cryptographic operation on low-power processors (e.g., ARM Cortex-M series). Public-key operations such as exponentiation are known to consume significantly more cycles than modular addition and lightweight hash computation. The model estimates the energy of one audit on an edge node as $E = P \cdot t + e \cdot B$, where t is the audit processing time (the verification time in Table 3), B is the number of bytes exchanged in the audit (Table 4, with 1 KB = 1,024 bytes), P = 0.5 W is the active power of the 300 MHz edge processor, and e = 1 µJ per byte is the radio transmission energy. For PDP at 10,000 blocks, t = 76 ms and B = 6.8 KB give E = 0.5 W × 76 ms + 1 µJ × 6,963 bytes ≈ 45.0 mJ. The same formula was applied to every scheme and dataset size, and the results are reported in millijoules per audit in Table 6. For 10,000 blocks, PDP required approximately 45.0 mJ and PoR about 52.8 mJ (owing to its longer processing time and larger responses). The proposed framework consumed roughly 24.0 mJ, a 47% reduction compared with PDP and a 55% reduction compared with PoR. At 100,000 blocks, PDP and PoR increased to 301.7 mJ and 360.5 mJ, respectively, while the proposed scheme required approximately 154.8 mJ (reductions of 49% and 57%), showing consistent efficiency gains as illustrated in Figure 6 and summarised in Table 6. The values are model-based estimates and were not measured on physical hardware.

Table 6. Estimated energy consumption per audit (mJ)

Dataset Size (Blocks)	PDP (mJ)	PoR (mJ)	Proposed (mJ)	Reduction vs PDP	Reduction vs PoR
10,000	45.0	52.8	24.0	47%	55%
50,000	164.9	198.0	83.2	50%	58%
100,000	301.7	360.5	154.8	49%	57%

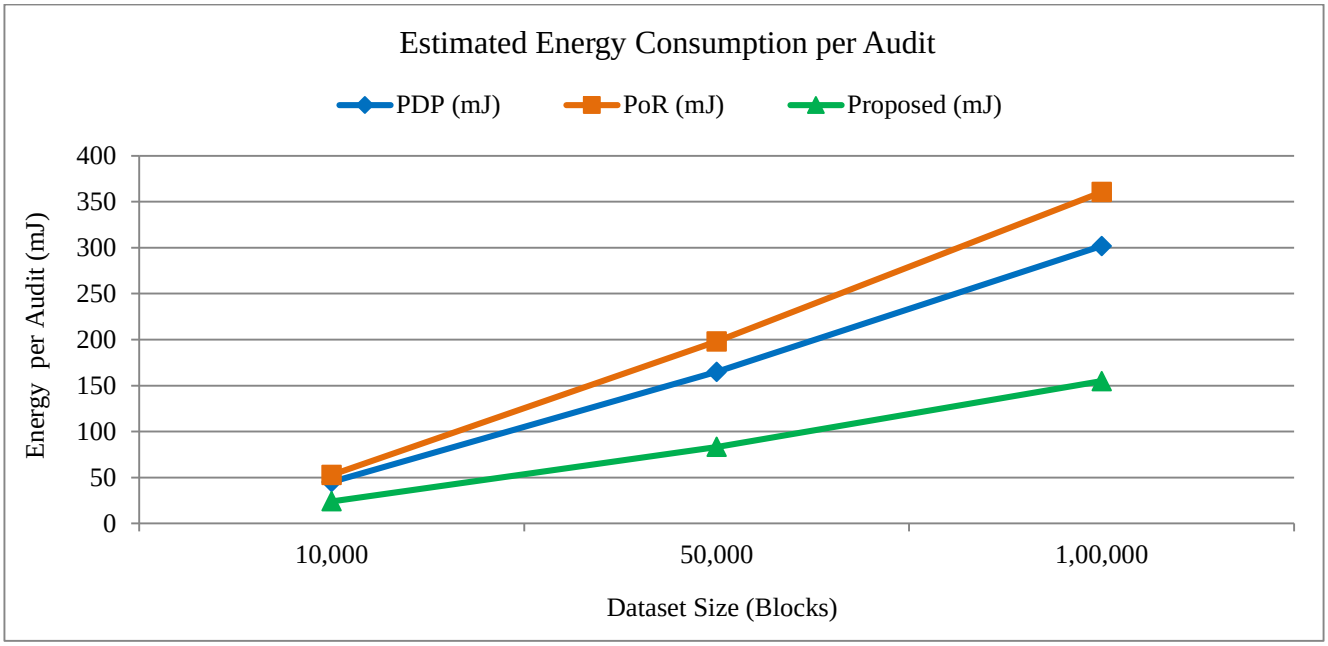

Fig. 6 Estimated energy consumption (mJ) vs dataset size

5.6. Scalability

In addition to the individual metrics, we also investigated scalability by increasing the dataset size. The verification time in the reported runs of all three schemes increased approximately linearly with the number of blocks, as did the proposed framework (from 41ms to 300ms), but at a lower rate: PDP (from 76ms to 584ms); PoR (from 89ms to 698ms). The proof size also increased, albeit at a lower rate, from 3.4 KB to 4.7 KB for a 10-fold increase in the number of blocks (Table 4); this is expected given the logarithmic length of the authentication path, following the same development as determined in Section 3.7. At 100,000 blocks (~1.6 GB at the average block size of 16KB), PDP required 584ms and PoR required 698ms to verify the blocks, compared with 300ms for the proposed framework. These results show that hierarchical aggregation does not scale as poorly on the large data sets prevalent in cloud storage systems as the baselines.

5.7. Statistical Validation

Each experiment was conducted in a controlled manner and repeated to ensure reliable, consistent results. To limit random fluctuations in system performance, each evaluation scenario was repeated 10 times with the same set of system parameters, the same setting of the data sets, and the same setting of the network; the presentation of the values presented in Tables 3–6 are expressed as the mean of the 10 runs.

Besides the mean, the standard deviation of each performance measure, namely computational cost, communication overhead, storage overhead, and energy

consumption, was also computed for the ten runs, and it was verified that the results were stable across multiple runs.

A two-sided paired t-test was used at the 95% confidence level (significance level $\alpha = 0.05$, the default value in experimental computer science) on the measurements performed per run for the proposed framework and each of the baselines (PDP, PoR) to check whether the improvements were significant or not. The test statistic for each metric is the mean of the 10 differences (per run) between the baseline and the proposed framework divided by the standard error of those differences, or their standard deviation divided by 10.

The test has 10 paired runs, meaning there are 9 degrees of freedom, and the two-sided critical value at $\alpha = 0.05$ is $t = 2.262$, so a difference was considered significant if $|t| > 2.262$ ($p < 0.05$). For both baselines, computation and communication costs were reduced by more than this margin, suggesting the benefits of the proposed framework were not due to random variability across multiple runs.

Such statistical validation adds credibility to the experimental results and supports the conclusion that the proposed integrity verification framework is effective in a cloud-edge computing environment.

5.8. Summary of Findings

Across all evaluated dimensions, the proposed framework consistently outperformed traditional PDP and PoR models. Quantitative improvements ranged from approximately 40–67 % depending on the metric, confirming that a lightweight hierarchical design is significantly more efficient in environments characterised by resource limitations and high-volume data streams.

Metrics such as reduced communication overhead and lower energy consumption are particularly critical for IoT and edge settings, where bandwidth and power are limited. The use of realistic telemetry data from the ToN-IoT dataset further strengthens the validity of these findings, showing that the framework is efficient not only in theory but also under realistic telemetry workloads; validation on physical edge hardware remains future work (Section 6.3).

5.9. Discussion of Performance Improvements over Existing Approaches

Comparisons with the selected PDP and PoR baselines, based on experimental results, show that the proposed Lightweight Hierarchical Merkle-Based Integrity Verification Framework reduces computation time, communication overhead, storage overhead, and estimated energy costs in the studied cloud–edge scenarios. This can be understood in the context of the hierarchical assignment of verification tasks, hash-based item identification and authentication-path information, and the summation of authentication-path information during batch verification. The extent of the enhancement, however, depends on the baseline construction used, how the workload is set up, challenge size, and measurement methods.

Computational efficiency. The proposed framework's proof-generation time varies from 94 ms to 712 ms as the number of blocks in the dataset increases, while PDP times range from 182 ms to 1,480 ms, and PoR times range from 215 ms to 1,732 ms. On the other hand, the proof times of the selected baselines have a higher range, as explained in Section 5.2. This improvement is mostly due to the shift from the more time-consuming authentication operations of these specific PDP and PoR implementations to the use of block authentication with the SHA-256 hash function and Merkle-tree verification. Also, a hierarchical architecture pushes tree construction and aggregation to the Gateway, rather than requiring every constrained edge device to perform the full verification workflow. As a result, the proposed design reduces the computational work for the evaluated audit operations. These observations are for the prototype that has been implemented and the baselines chosen; they do not apply to each PDP/PoR built.

Communication efficiency. From the results presented in Section 5.3, it is observed that the proposed framework can overcome a communication overhead of 3.4KB, 4.1KB and 4.7KB for datasets of 10000, 50000 and 100000 blocks, respectively. These are lower than the corresponding PDP and PoR baselines reported. The ultimate architectural rationale is that the verifier knows the challenged "leaf values" and the authentication information only to recover the Merkle root that was committed, and not download the whole outsourced data for each audit. Moreover, authentication-path nodes common to the v leaves being challenged are sent only once. This minimises the re-send of verification metadata for in-batch verification. The amount of communication that can be saved is important in an edge environment where bandwidth is scarce in nature but depends on the size of the audit and the rules about communication accounting that may be enforced for each baseline.

Storage efficiency. The proposed framework was shown to have a 6.2% metadata storage overhead, while PDP has 10.4% and PoR has 18.6% as reported in Section 5.4. To minimise authentication metadata, the proposed design avoids the extra redundancy of the chosen PoR implementation. These characteristics can reduce the amount of verification information that must be stored. However, Merkle-tree authentication is not "storage free": consider hashes of the leaves, internal nodes, and version information, as well as other persistent metadata. Hence, interpret the reported storage advantage in terms of the storage-overhead definition and accounting boundaries defined in the experimental setup. Note that this comparison is not meant to imply that all PoR constructions need the same amount of redundancy or that all PDP constructions have a metadata requirement that is the same.

Estimated energy efficiency. Section 5.5 assumes the processing-time and the communication-volume model as defined in the experimental methodology to estimate energy consumption. With this model, shorter processing time and fewer transmitted bytes both help lower the estimated

energy an audit consumes. The potential advantages of the proposed framework are that of having the lower computational requirements of the hash-based audit, and the lower volume of communications being reported. The energy measurements thus align with the computational and communication trends established in the experiments. However, these are only model values and are never measured on real physical edge devices. This means the results should be viewed as an indication of possible energy savings under the conditions mentioned.

Comparative evaluation and the results, in general. Overall, the findings indicate that the proposed framework's main performance gain comes from coordinating lightweight block hashing, gateway-assisted Merkle-tree construction, and batch authentication-path verification. These components solve the different overheads: hashing reduces the computation cost of block authentication, hierarchical processing distributes work according to the capabilities of the system components, and shared authentication paths reduce repeated communication during audits. The selected experiments support the effectiveness of this combination within the evaluated workload range of 1,000 to 100,000 blocks. Nevertheless, the comparison should be interpreted in light of the specific baseline implementations and the security properties evaluated. In particular, the measured prototype implements the sampled Merkle audit, whereas the hardened ring-SIS aggregate and device-level range audit are analysed separately and were not included in the measured experiments. Moreover, the sampled Merkle audit is consistent with an authenticated commitment under the stated assumptions; by itself, it does not establish data possession or retrievability equivalent to a complete PDP or PoR guarantee. Further validation using physical edge devices, fully specified baseline implementations, and measurements of the additional proposed components would strengthen the evidence for practical deployment.

6. Conclusion

This paper introduced a Lightweight Hierarchical Data Integrity Check Framework that is meant to assist edge devices that are resource-limited and which are deployed in cloud-integrated systems. Classical integrity verification methods, including Provable Data Possession (PDP) and Proof of Retrievability (PoR), rely on computationally expensive public-key cryptographic operations and redundancy checks that are incompatible with the processing and memory constraints of IoT-based systems. To address these constraints, the proposed framework incorporates lightweight homomorphic hashing, hierarchical Merkle tree aggregation, and a batch auditing system to enable smooth, scalable integrity verification.

The suggested architecture allocates verification tasks across layers of the cloud-edge ecosystem. Lightweight hashing is done by the edge devices, whereas more computationally intensive aggregation is done by the intermediate edge gateways. This top-down architecture minimises the amount of computation that must be done on the device side, as well as minimising the cost of communication between system elements. Experimental analysis shows that the framework is more efficient than conventional integrity verification frameworks. The findings showed significant decreases in computational, communication, storage, and estimated energy costs, indicating the adequacy of the proposed strategy for massive IoT and edge computing systems.

In addition to the empirical results, the paper formally defined an additive block hash, the Merkle commitment, and a position-keyed ring-SIS aggregate (Section 3). It proved that an accepted proof is a binding one even if tampering of committed blocks and/or tree nodes (Theorems 1 and 2), that a fraction ρ of blocks are corrupted and is detectable with a probability at least $1 - (1 - \rho)k$ if k blocks are challenged (Theorem 3), that replayed or rolled-back data are rejected (Theorem 4), that the hardened aggregate is binding under the ring-SIS assumption (Theorem 5), and that a colluding gateway and cloud server cannot present committed modified blocks as the data of the device during a range audit (Theorem 6). These guarantees hold under the aforementioned assumptions about authenticated channels and proofs derived from stored blocks.

6.1. Research Implications

The results of the study bear significant implications for the design of safe data management systems within cloud-edge and IoT systems. The study offers a practical solution to the challenge of limited computational resources by showing that, in resource-constrained environments, lightweight hashing and hierarchical aggregation can provide reliable integrity verification without heavy cryptographic primitives. The suggested framework also facilitates the scalability of data validation in distributed systems that produce large quantities of data, including smart city monitoring systems, industrial IoT infrastructures, and environmental observation systems. In addition, the hierarchical verification model fits well with modern edge computing systems in which the gateway nodes may help in achieving computationally demanding tasks, which enhance the overall efficiency and reliability of the system.

6.2. Limitations

Although the experimental assessment showed performance gains, several limitations remain. First, the suggested framework was considered mostly in a cloud-edge setting, which may not reflect the complexity and diversity of real deployments. Second, the existing implementation primarily verifies static data and does not scale well to situations involving regular dynamic updates, e.g., inserting, deleting, or changing a block. Third, although the framework minimises computational requirements, it lacks more sophisticated privacy-preserving audit protocols, which may be required when the verification procedure is outsourced to other organisations. Fourth, the security analysis considers the proofs to be computed on the blocks stored and, like other hash-based schemes, it authenticates content commitment but not possession or retrievability

against a server that stores the hashes and forgets the content. Fifth, the hardening ring-SIS aggregate and the range audit are analysed, even though the experiments were not measured; the system parameters should be compared with lattice-reduction estimators. Sixth, the power values of the obtained figures are also estimated by an operation-count model and given device parameters, and not measured on real devices. These drawbacks should be remediated for the model to become more viable and solid.

6.3. Future Research Directions

The suggested framework can be developed in a number of directions in future research. A possible improvement includes the implementation of blockchain-based distributed auditing [24, 25] that can enhance the level of transparency and resistance to tampering in the process of integrity checking. The other future opportunity is the use of privacy-preserving verification methods, e.g., zero-knowledge proofs and secure multi-party computation, to secure sensitive metadata when auditing. Also, future research can be conducted to test the offered framework applied to actual IoT hardware platforms to understand how it performs in real-life deployment conditions.

Another direction is to combine the block hash with keyed homomorphic tags verified by the data owner, in the style of the private-verification PoR scheme [9], which would remove the need to assume proofs are generated from the stored blocks, at the cost of larger proofs. Finally, incorporating dynamic data update schemes and adaptive verification schemes may also improve the scalability and efficiency of large-scale cloud-edge systems.

Funding

The authors received no specific funding for this research.

Institutional Review Board Statement

Not applicable.

Authors' Contributions

N. Musrat Sultana contributed to the conceptualization and design of the study, carried out the experimental work, performed data analysis and interpretation, and drafted the manuscript. The author critically reviewed and approved the final version of the manuscript.

Disclosure of AI Use

The author used OpenAI's ChatGPT to assist with language editing and improving the clarity of certain sections of the manuscript. All AI-generated content was carefully reviewed, verified, and revised by the author to ensure the accuracy, originality, and integrity of the work.

Citation Policy

The authors confirm that all citations in this manuscript are relevant, balanced, and directly related to the content of the study. Care has been taken to avoid excessive self-citation or disproportionate referencing of work from the authors' institution or collaborators. Unrelated or redundant references have been removed to maintain clarity and objectivity. The authors affirm that no inappropriate or misleading citations have been included and that the manuscript complies fully with the journal's citation ethics policy.

References

[1] Michael Armbrust et al., "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[2] Rajkumar Buyya et al., "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599-616, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[3] Peter Mell, and Tim Grance, "The NIST Definition of Cloud Computing," *Computer Security Resource Center*, 2011. [Google Scholar] [Publisher Link]

[4] Mahadev Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30-39, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[5] Holger Karl, and Andreas Willig, *Protocols and Architectures for Wireless Sensor Networks*, vol. 1, Wiley, NY, 2005. [CrossRef] [Google Scholar] [Publisher Link]

[6] S. Subashini, and V. Kavitha, "A Survey on Security Issues in Cloud Computing," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 1-11, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[7] Giuseppe Ateniese et al., "Provable Data Possession at Untrusted Stores," *Proceedings of the 14th ACM Conference on Computer and Communications Security*, Association for Computing Machinery, New York, United States, pp. 598-609, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[8] A. Juels and B. Kaliski, "Proofs of Retrievability for Large Files," *Proceedings of the 14th ACM conference on Computer and Communications Security*, Association for Computing Machinery, New York, United States, pp. 584-597, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[9] Hovav Shacham, and Brent Waters, "Compact Proofs of Retrievability," *Journal of Cryptology*, vol. 26, no. 3, pp. 442-483, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[10] Cong Wang et al., "Privacy-Preserving Public Auditing for Secure Cloud Storage," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362-375, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[11] Kan Yang, and Xiaohua Jia, "An Efficient and Secure Dynamic Auditing Protocol for Data Storage in Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1717-1726, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[12] Yan Zhu et al., "Dynamic Audit Services for Outsourced Storages in Clouds," *IEEE Transactions on Services Computing*, vol. 6, no. 2, pp. 227-238, 2011. [CrossRef] [Google Scholar] [Publisher Link]
[13] Hui Tian et al., "Dynamic-Hash-Table based Public Auditing for Secure Cloud Storage," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 701-714, 2015. [CrossRef] [Google Scholar] [Publisher Link]
[14] Hongliang Zhu et al., "A Secure and Efficient Data Integrity Verification Scheme for Cloud-IoT based on Short Signature," *IEEE Access*, vol. 7, pp. 90036-90044, 2019. [CrossRef] [Google Scholar] [Publisher Link]
[15] Neenu Garg, and Seema Bawa, "RITS-MHT: Relative Indexed and Time Stamped Merkle Hash Tree based Data Auditing Protocol for Cloud Computing," *Journal of Network and Computer Applications*, vol. 84, pp. 1-13, 2017. [CrossRef] [Google Scholar] [Publisher Link]
[16] Ralph C. Merkle, "Protocols for Public Key Cryptosystems," *1980 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 1980.
[CrossRef] [Google Scholar] [Publisher Link]
[17] C. Chris Erway et al., "Dynamic Provable Data Possession," *ACM Transactions on Information and System Security*, vol 17, no. 4, pp. 1-29, 2009. [CrossRef] [Google Scholar] [Publisher Link]
[18] Whitfield Diffie, and Martin E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. 22, vo. 6, pp. 644-654, 1976. [CrossRef] [Google Scholar] [Publisher Link]
[19] Ronald L. Rivest, Adi Shamir, and Len Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 26, no. 1, pp. 96-99, 1978. [CrossRef] [Google Scholar] [Publisher Link]
[20] Dan Boneh, and Matt Franklin, "Identity-based Encryption from the Weil Pairing," *Advances in Cryptology - CRYPTO 2001 21st Annual International Cryptology Conference*, Santa Barbara, California, USA, vol. 2139, pp. 213-229, 2003. [CrossRef] [Google Scholar] [Publisher Link]
[21] Paromita Goswami et al., "Investigation on Storage Level Data Integrity Strategies in Cloud Computing," *Journal of Cloud Computing*, vol. 13, no. 1, pp. 1-23, 2024. [CrossRef] [Google Scholar] [Publisher Link]
[22] Yan Ding et al., "Edge Data Integrity Verification Scheme Supporting Data Dynamics and Batch Auditing," *Journal of Systems Architecture*, vol. 128, pp. 1-12, 2022. [CrossRef] [Google Scholar] [Publisher Link]
[23] Liping Qiao et al., "Lightweight Integrity Auditing of Edge Data for Distributed Edge Computing Scenarios," *Ad Hoc Networks*, vol. 133, pp. 1-11, 2022. [CrossRef] [Google Scholar] [Publisher Link]
[24] Bin Liu et al., "Blockchain Based Data Integrity Service Framework for IoT Data," *2017 IEEE International Conference on Web Services (ICWS)*, Honolulu, HI, USA, pp. 468-475, 2022. [CrossRef] [Google Scholar] [Publisher Link]
[25] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *SSRN*, pp. 1-9, 2008. [Google Scholar] [Publisher Link]
[26] Jiale Zhang et al., "LVPDA: A Lightweight and Verifiable Privacy-Preserving Data Aggregation Scheme for Edge-Enabled IoT," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4016-4027, 2023. [CrossRef] [Google Scholar] [Publisher Link]
[27] Haiyan Wang, Yi Lin, and Fu Xiao, "A Lightweight Data Integrity Verification with Data Dynamics for Mobile Edge Computing," *Security and Communication Networks*, vol. 2022, 2022. [CrossRef] [Google Scholar] [Publisher Link]
[28] Fahad F. Alruwaili, "Ensuring Data Integrity in Deep Learning-Assisted IoT-Cloud Environments: Blockchain-assisted Data Edge Verification with Consensus Algorithms," *AIMS Mathematics*, vol. 9, no. 4, pp. 8868-8884, 2024. [CrossRef] [Google Scholar] [Publisher Link]
[29] Yao Zhao et al., "Data Integrity Verification in Mobile Edge Computing with Multi-Vendor and Multi-Server," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 5418-5432, 2024. [CrossRef] [Google Scholar] [Publisher Link]
[30] Jun Ye, and Yu Jiang, "Data Integrity Verification for Edge Computing Environments," *Symmetry*, vol. 16, no. 12, pp. 1-20, 2024. [CrossRef] [Google Scholar] [Publisher Link]
[31] Zahra Seyedi et al., "Decentralized Data Integrity Inspection Offloading in Edge Computing," *IEEE Transactions on Mobile Computing*, vol. 24, no. 5, pp. 3950-3961, 2025. [CrossRef] [Google Scholar] [Publisher Link]
[32] Yao Zhao et al., "A Comprehensive Survey on Edge Data Integrity Verification: Fundamentals and Future Trends," *ACM Computing Surveys*, vol. 57, no. 1, pp. 1-34, 2024. [CrossRef] [Google Scholar] [Publisher Link]
[33] He Xue et al., "Integration of Blockchain and Edge Computing in Internet of Things: A Survey," *Future Generation Computer Systems*, vol. 144, pp. 307-326, 2022. [CrossRef] [Google Scholar] [Publisher Link]
[34] Peiying Zhang et al., "A Reliable Data-Transmission Mechanism using Blockchain in Edge Computing Scenarios," *IEEE Internet of Things Journal*, vol. 9, no. 16, pp. 14228-14236, 2022. [CrossRef] [Google Scholar] [Publisher Link]
[35] Gunasekaran Manogaran et al., "Blockchain Assisted Secure Data Sharing Model for Internet of Things based Smart Industries," *IEEE Transactions on Reliability*, vol. 71, no. 1, pp. 348-358, 2021. [CrossRef] [Google Scholar] [Publisher Link]
[36] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser, "Incremental Cryptography: The Case of Hashing and Signing," *Advances in Cryptology – CRYPTO '94*, Berlin, Heidelberg: Springer, vol. 839, pp. 216-233, 1994. [CrossRef] [Google Scholar] [Publisher Link]
[37] David Wagner, "A Generalized Birthday Problem," *Annual International Cryptology Conference*, Santa Barbara, CA, USA, pp. 288-304, 2002. [CrossRef] [Google Scholar] [Publisher Link]
[38] Torben Pryds Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," *Advances in Cryptology – CRYPTO '91*, Berlin, Heidelberg: Springer, vol. 576, pp. 129-140, 1991. [CrossRef] [Google Scholar] [Publisher Link]

[39] Mikols Ajtai, "Generating Hard Instances of Lattice Problems," *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York, United States, pp. 99-108, 1996. [CrossRef] [Google Scholar] [Publisher Link]

[40] Vadim Lyubashevsky, and Daniele Micciancio, "Generalized Compact Knapsacks are Collision Resistant," *Automata, Languages and Programming (ICALP 2006)*, Berlin, Heidelberg: Springer, pp. 144-155, 2006. [CrossRef] [Google Scholar] [Publisher Link]

[41] Vadim Lyubashevsky et al., "SWIFFT: A Modest Proposal for FFT Hashing," *Fast Software Encryption (FSE 2008)*, Berlin, Heidelberg: Springer, pp. 54-72, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[42] Abdullah Alsaedi et al., "TON_IoT Telemetry Dataset: A New Generation Dataset of IoT and IIoT for Data-Driven Intrusion Detection Systems," *IEEE Access*, vol. 8, pp. 165130-165150, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[43] N. Musrat Sultana, and K. Srinivas, "Data Privacy Protection in Cloud Computing using Visual Cryptography," *Multimedia Tools and Applications*, vol. 84, no. 21, pp. 23577-23597, 2024. [CrossRef] [Google Scholar] [Publisher Link]