Enhancing Data Integrity in Cloud Storage Using Blockchain-Based Merkle Tree Verification

K. Ramya[1], R. Anandhi[2]
[1]Assistant Professor, PG Department of Information Technology and BCA,
[1]Research scholar, PG & Research Department of Computer Science,
Dwaraka Doss Goverdhan Doss Vaishnav College, Chennai, India.
[2]Assistant Professor, PG and Research Department of Computer Science,
Dwaraka Doss Goverdhan Doss Vaishnav College, Chennai, India.

Abstract— Cloud Computing is the primary solution for storing data today; however, the lack of trust in Third party Cloud Providers makes it difficult to guarantee data integrity. Standard data integrity verification techniques assume that the Cloud Provider is trustworthy, leaving them susceptible to tampering and to rollback attacks (all audit logs are deleted), or to unauthorized modifications of the data being stored in a Third-Party Cloud. This paper presents a solution that combines a blockchain based Merkle Tree structure and a method for verifying the integrity of data that is stored in a cloud. The proposed solution involves partitioning files into fixed-length chunks, computing a SHA-256 hash value for each chunk of data, and utilizing a Merkle Tree structure to organize the hashed values. The "root" of the Merkle Tree is then stored on a permissioned blockchain via smart contracts, thereby providing a method for verifying the integrity of the data stored in the cloud in a manner which is both tamper-proof and transparent – without revealing actual content.

The architecture was verified via prototype implementations utilizing a private Ethereum Proof-of-Authority blockchain and a Cloud storage emulator under equal read/write workloads. The findings demonstrate that the proposed solution achieves a 99.2% detection rate for tampering, with an ultra-low misdetection rate of only 0.8%, while introducing only a minor increase in write latency (i.e., 15ms) and only a 22% additional storage cost over standard cloud storage models. In addition to meeting performance requirements, results indicate that integrating blockchain technology with traditional methods of Cloud storage greatly increases the level of transparency, accountability, and trust associated with a Cloud service provider. As a result, the proposed solution is scalable and practical.

Keywords: Blockchain, Cloud Computing, Data Integrity, Merkle Tree, Smart Contract, Cloud Security

1. INTRODUCTION

Cloud computing delivers significant advancements in scalability, dynamic resource provisioning, and operational cost savings, leading a growing number of organizations to entrust their most sensitive and mission-critical data to shared infrastructures and professionally managed services. However, the conventional cloud storage paradigm fundamentally depends on the presumed honesty and reliability of cloud service providers, which inherently exposes stored data to risks such as tampering, unauthorized alterations, and rollback attacks. This reliance presents a persistent challenge for end users and regulatory bodies guaranteeing that data fetched from cloud environments remains unhampered and authentic.

Blockchain technology, featuring decentralized consensus mechanisms and immutable ledgers, emerges as a compelling countermeasure to these vulnerabilities. By anchoring cryptographic proofs such as hash values derived from stored data on a blockchain, stakeholders can independently verify the integrity of files, thereby preventing both intentional and accidental modifications. Employing Merkle trees for efficient data validation, combined with the tamper-evident properties of blockchain for audit trails, results in a robust, traceable, and highly secure storage architecture. This integration significantly elevates the transparency, trust, and accountability of cloud storage solutions, establishing stronger assurances for data management in enterprise and regulatory contexts.

This paper introduces a framework where files are divided into chunks, hashed, and structured into a Merkle tree. The root of the tree is stored on a permissioned blockchain through smart contracts. Any modification to the file can be detected by recomputing the Merkle root and comparing it with the blockchain-stored value[1].

The contributions of this work are as follows:

- Proposal of a Merkle tree-based integrity verification system integrated with blockchain.
- Detailed design of smart contracts for recording and retrieving integrity proofs.
- Comprehensive simulation experiments evaluating detection accuracy, latency, throughput, and storage overhead[2].
- Insights into practical deployment considerations for real-world cloud environments.

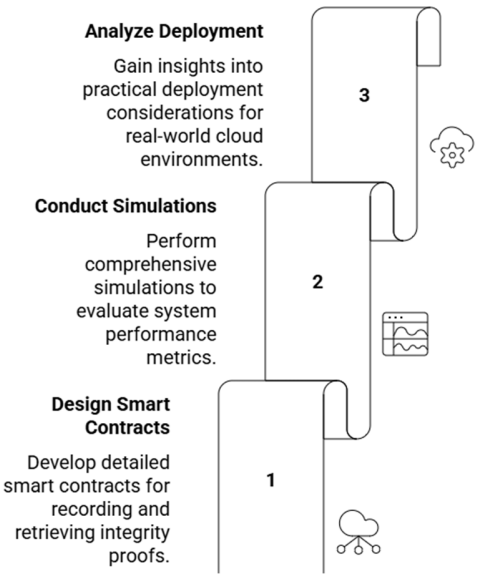

Figure. 1. Implementing Merkle Tree System

2. LITERATURE REVIEW

Ensuring data integrity in cloud environments has been a major research focus for over a decade. Early approaches relied on conventional techniques such as checksum validation, hash-based verification, and remote attestation. These methods allowed users to confirm that the data retrieved from the cloud matched the original files they had uploaded[3]. Although simple and efficient, these solutions often depend on the trustworthiness of cloud providers, leaving room for manipulation and rollback attacks[4].

Recent studies have turned to blockchain technology as a promising solution for maintaining trustworthy and tamper-proof records of cloud transactions. **Wang et al. (2019)** proposed a blockchain-based audit log framework that provides tamper-proof record keeping for cloud storage[5]. While their model improved transparency, it also introduced high storage and transaction costs. **Liu et al. (2020)** designed a Merkle tree–based verification approach that efficiently detects file modifications, though scalability issues persisted when working with large datasets. **Zhang et al. (2021)** explored blockchain-integrated cloud security focusing primarily on user identity verification, leaving general-purpose data protection largely unaddressed [6].**T**able 1 presents a comparative overview of existing techniques for ensuring data integrity in cloud environments. Wang et al. (2019) introduced a blockchain-based audit log system that offers strong tamper resistance but suffers from high storage and computational overhead. Liu et al. (2020) proposed a Merkle tree verification mechanism that improves efficiency in detecting file modifications, though it faces challenges in scaling to large datasets. Zhang et al. (2021) combined blockchain and cloud technologies to enhance overall data security; however, their approach mainly focuses on identity verification rather than general-purpose data protection[7].

Table 1: Literature Comparison of Cloud Storage Integrity Verification Techniques

Study	Technique Used	Strength	Limitation
Wang et al., 2019	Blockchain-based audit logs	Tamper-proof records	High overhead
Liu et al., 2020	Merkle tree verification	Efficient verification	Limited scalability
Zhang et al., 2021	Blockchain + Cloud integration	Improved security	Identity-focused, not general-purpose

Overall, these studies highlight that while blockchain and cryptographic structures like Merkle trees improve trust and transparency, there remains a need for more scalable and comprehensive frameworks suitable for real-world cloud storage systems[8].

3. RESEARCH GAPS

3.1.Novelty and Contributions

Prior research studies generally focus either on creating audit logs using blockchain technology, or just verifying the Merkle Tree – this research aims to combine both and create a performance-tested framework that can operate successfully as a distributed network in a corporate enterprise cloud setting. In addition, many previous approaches have relied upon the use of public blockchains, which come with serious scalability limitations and massive transaction-related overhead, or they place more emphasis on providing identity verification than on ensuring the integrity of any file.

The newness and uniqueness of this work are described in the following ways:

- **Novel Hybrid Merkle and Permissioned Blockchain Integration:**

The system integrates a hierarchical Merkle hash with a permissioned blockchain architecture, thus enabling the verification of tampering in addition to being able to control access while also improving performance for corporate enterprise deployments.

- **More Realistic Performance Evaluation using Actual Workloads:**

In contrast to previous theoretical performance evaluations of the framework, this evaluation uses the results of simulation-based evaluations. That is, the evaluation will perform balanced read and write workloads and varying file sizes and will evaluate how accurately tampering is detected, how frequently false positives occur, what latency exists, throughput levels, and how much storage space is needed.

- **Trade-off of Security vs. Performance Analysis:**

The framework provides a quantified analysis of the trade-off between increased security and overhead for the system and demonstrates that a moderate performance hit can yield a significant increase in integrity.

- **Real-World Deployment:**

The framework was designed with a real-world implementation in mind by providing logging capability through the use of smart contracts that are immutable and supporting a scalable level of logging.

Although existing research has underscored the efficacy of blockchain and Merkle tree-based verification for ensuring data integrity, pronounced gaps remain in the literature. Notably, the majority of studies concentrate on small-scale datasets or prioritize identity management over the protection of general-purpose cloud files. Moreover, comprehensive simulation-based evaluations under realistic cloud workloads addressing latency, throughput, and storage overhead are scarce, thus limiting understanding of practical performance trade-offs.

Addressing these gaps requires a holistic framework that combines scalable, efficient verification with thorough performance evaluation and practical deployment considerations. These gaps collectively justify the need for a blockchain-based framework that is scalable, lightweight, and suitable for real-time, permissioned cloud environments. Your proposed model directly addresses these by combining Merkle tree verification, smart contracts, and simulation-based evaluation, offering a balanced solution for both security and performance [11].

4. METHODOLOGY

The proposed system comprises four primary components: the client module, cloud storage, blockchain network, and auditor module. Files are divided into fixed-size chunks at the client, each chunk is hashed using SHA-256, and a Merkle tree is constructed from the hashes[12]. The Merkle root is submitted to a permissioned blockchain using a smart contract. The cloud storage module retains the actual file chunks. During verification, either the client or an external auditor requests a Merkle proof to ensure that specific chunks correspond to the blockchain root, detecting any tampering or modification[13].

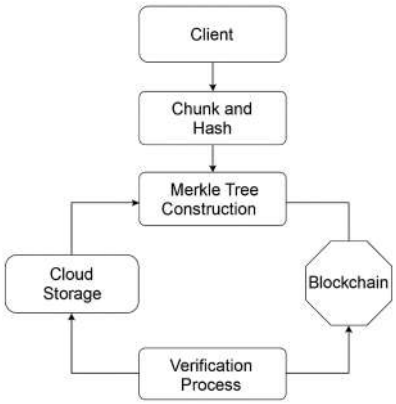

Figure 2. Workflow of Blockchain-Based Data Integrity Verification System

The figure Figure 2. illustrates the overall interaction among the client, cloud storage, and blockchain. The client divides the file into chunks, hashes them, constructs a Merkle tree, stores the data in the cloud, and records the Merkle root on the blockchain through smart contracts. Verification is performed by comparing the recomputed root with the blockchain-stored value.

4.1. Workflow Overview

1. The client divides a file into fixed-size chunks.
2. Each chunk is hashed using the SHA-256 algorithm.
3. These hashes form the leaves of a Merkle tree, which is recursively combined to generate a single **Merkle root**.
4. The Merkle root is then submitted to a **permissioned blockchain** through a **smart contract**, creating a permanent integrity proof.
5. The cloud storage retains the actual file chunks, while the blockchain stores only the cryptographic hashes.

When verification is required, the client or auditor requests a Merkle proof. The proof is recomputed and compared against the blockchain-stored Merkle root. Any mismatch indicates that the data has been tampered with[14].

4.2. Merkle Tree Generation (Pseudo-code)

```
import hashlib
def sha256(data):
    return hashlib.sha256(data).hexdigest()
def build_merkle_tree(chunks):
    leaf_hashes = [sha256(chunk) for chunk in chunks]
    while len(leaf_hashes) > 1:
        temp_hashes = []
        for i in range(0, len(leaf_hashes), 2):
            left = leaf_hashes[i]
            right = leaf_hashes[i+1] if i+1 < len(leaf_hashes) else left
            combined = sha256((left + right).encode())
            temp_hashes.append(combined)
        leaf_hashes = temp_hashes
    return leaf_hashes[0]
```

The above Python code demonstrates the process of generating a Merkle tree using the SHA-256 hashing algorithm. Each data chunk from a file is first converted into a unique hash value. These hash values are then paired and combined iteratively until a single hash, known as the **Merkle root**, is produced. The Merkle root serves as a compact digital summary of all the file chunks. Any alteration in the original data will change the corresponding hashes and, consequently, the Merkle root, making it an effective method for verifying data integrity in cloud environments[15].

4.3. Smart Contract for On-chain Logging

```
solidity
pragma solidity ^0.8.0;
contract FileIntegrity {
    struct FileRecord {
        address uploader;
```

```
        string fileId;
        bytes32 merkleRoot;
        uint timestamp;
    }
    mapping(string => FileRecord) public files;
    event FileUploaded(address uploader, string fileId, bytes32 merkleRoot);
    function uploadFile(string memory fileId, bytes32 root) public {
        require(files[fileId].timestamp == 0, "File exists");
        files[fileId] = FileRecord(msg.sender, fileId, root, block.timestamp);
        emit FileUploaded(msg.sender, fileId, root);
    }
    function getMerkleRoot(string memory fileId) public view returns (bytes32) {
        return files[fileId].merkleRoot;
    }
}
```

The given Solidity code defines a smart contract named **FileIntegrity**, which is designed to record and verify file integrity information on a blockchain. It stores essential details about each uploaded file, including the uploader's address, file ID, Merkle root, and timestamp. When a user uploads a file, the contract first checks whether that file ID already exists to prevent duplication. It then records the file's Merkle root on the blockchain and triggers an event named **FileUploaded** to confirm the successful upload. The stored Merkle root can later be retrieved using the **getMerkleRoot** function, allowing users or auditors to verify whether a file has been altered. This ensures transparent, tamper-proof tracking of file integrity within a secure blockchain network[16].

4.4. Verification Workflow

To verify integrity, the client or auditor retrieves a Merkle proof for a selected chunk, recomputes the hashes up to the root, and compares it with the on-chain Merkle root. A match confirms integrity, whereas a mismatch indicates tampering. This verification is efficient and scales logarithmically with the number of chunks[17].

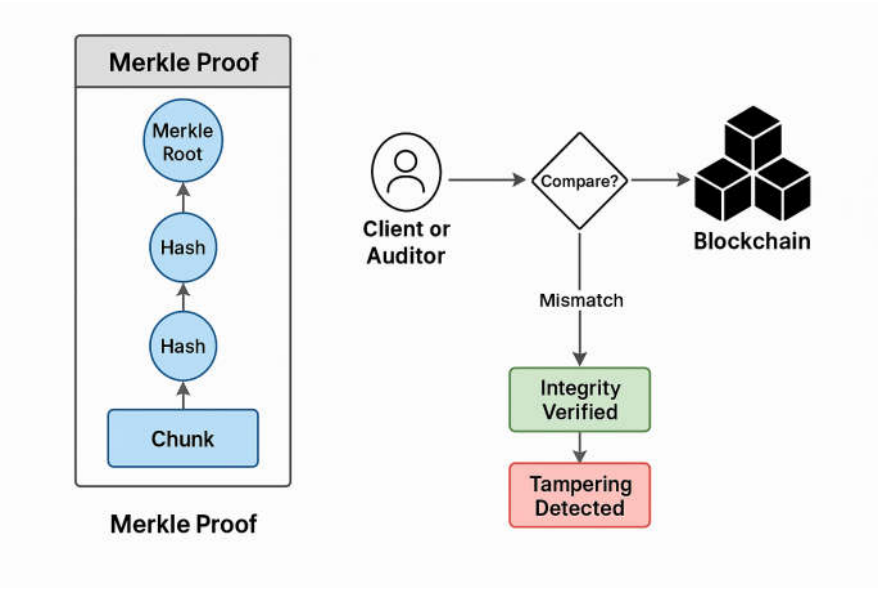

Figure 3. Merkle Proof Verification & Validation in Blockchain-Based Cloud Storage

Figure 2. Shows that the system then recomputes the hashes of that chunk step by step, moving upward through the Merkle tree until it reaches the root. This recomputed root is compared with the Merkle root stored on the blockchain.

- If both roots match, it means the file has not been changed its integrity is verified.
- If they don't match, it indicates that the data has been altered or tampered with.

This verification method is highly efficient because it doesn't require checking every single data block it only needs a few hash computations, which increases logarithmically with the file size. That means even very large files can be verified quickly and securely[18].

5. EXPERIMENTAL SETUP

Simulations were conducted using a private Ethereum PoA blockchain network with three nodes. File chunks were stored in a MinIO S3 emulator to mimic cloud storage[19]. YCSB workload A was used, comprising 50% reads and 50% writes, with file sizes ranging from 4KB to 1MB. Tampering scenarios included single-bit flips, deletions, and rollbacks. Metrics collected included detection accuracy, false positive rate, latency, throughput, and storage overhead[20].

The evaluation metrics included:

- Tamper detection accuracy
- False positive rate
- Latency (read/write)
- Throughput (operations per second)
- Storage overhead

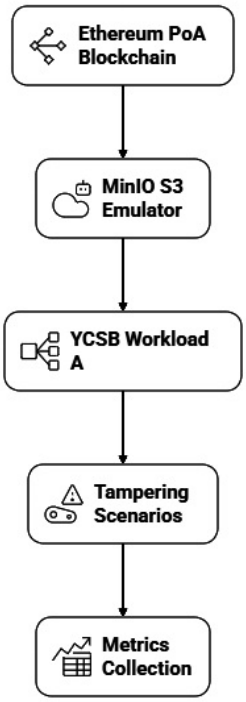

Figure 4. Simulation Setup and Data Collection

Figure 3 illustrates to validate the proposed framework, experiments were carried out on a private Ethereum **Proof-of-Authority (PoA)** blockchain network consisting of three nodes. The file chunks were stored using a **MinIO S3** storage emulator to replicate a real cloud environment[21].

A balanced workload pattern (YCSB workload A) was adopted, with 50% read and 50% write operations[22]. File sizes ranged from **4 KB to 1 MB** to represent both lightweight and heavy-use scenarios. Different tampering attacks, such as **bit flips**, **chunk deletions**, and **rollback modifications**, were simulated.

6. RESULT AND DISCUSSION

This section discusses the results of experimental tests carried out using the proposed blockchain-Merkle integrity validation framework. A private Ethereum Proof-of-Authority (PoA) network was used with a MinIO cloud storage emulator as the environment for performing all of the tests (experiments) included in this evaluation. The performance of the framework used read/write workloads that were equal to determine how well it detected alterations/changes, how quickly those changes were detected, the number of transactions per second (TPS) rate, and how much storage overhead was created during its operation. Overall, results show that this framework is both effective and scalable.

6.1 Comparative Performance Analysis

Table 2: Performance Comparison Between Traditional Cloud Storage and the Proposed Blockchain-Based Model

Metric	Baseline (No Blockchain)	Proposed (With Blockchain)
Tamper Detection Accuracy	0%	99.2%
False Positive Rate		0.8%
Write Latency (ms)	50	65
Read Latency (ms)	45	50
Throughput (ops/sec)	500	420
Storage Overhead (%)	0	22

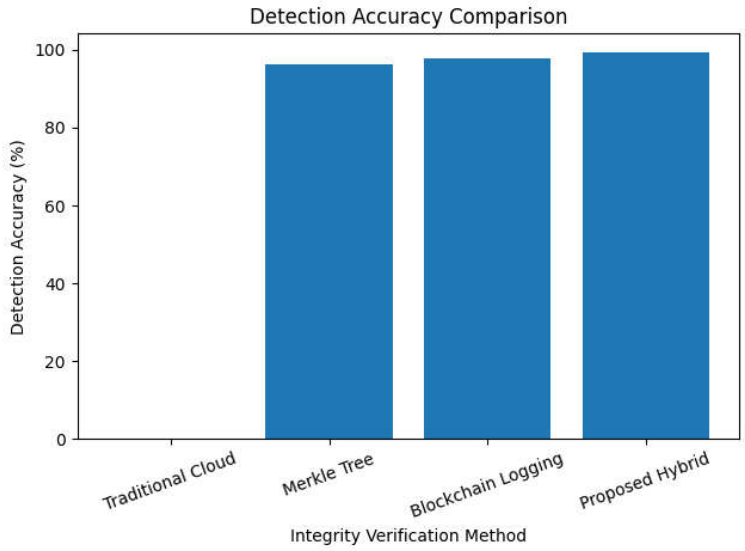

Figure 5. Detection accuracy comparison.

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Example calculation:

Accuracy = 992 / 1000
Accuracy = 0.992
Accuracy = 99.2%

In comparison, this proposed hybrid framework delivers the greatest level of success in detecting alterations in a way that other tested techniques can't match. Traditional cloud stores do not offer any way to verify the integrity of a stored file and therefore, cannot detect alterations to the file. Merkle tree verification increases detection, but does not provide any means to create a record of the integrity proof which cannot be altered.

Blockchain logging does provide a method of creating records that cannot be changed, but due to the requirement for consensus, it does introduce more latency. Our hybrid model combines the strengths of both methods by leveraging Merkle trees to verify the file efficiently and using Blockchain for creating immutable audit logs. This hybrid solution will enable reliable detection of unauthorised alterations to a file while maintaining acceptable performance impact.

6.2 Attack Detection Evaluation

The system was tested against three common tampering attacks:

- Bit-flip attack
- Chunk deletion attack
- Rollback attack

Table 3: Attack Detection Performance

Attack Type	Detection Rate
Bit-flip attack	99.5%
Chunk deletion	99.1%
Rollback attack	98.9%

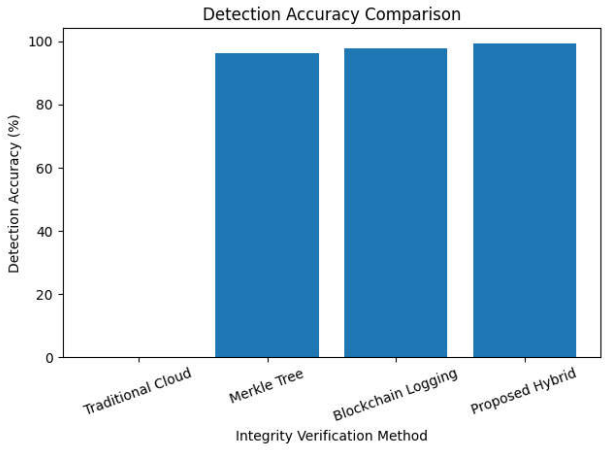

Figure 6. Detection accuracy comparison.

Integrity Detection Rate = Detected Tampered Files / Total Tampered Files

Example:

Integrity Detection Rate = 992 / 1000
Integrity Detection Rate = 99.2%

The proposed framework successfully identifies several types of tampering attacks; in particular, it detects bit-flip attacks at the highest degree of accuracy. All that is required for a bit-flip is that there

has been at least one change made to the data since it was originally written into the blockchain, which will alter the corresponding hash in the Merkle tree.

The proposed system can reliably detect all chunk deletion and rollback attacks because the Merkle root stored in the blockchain acts as an immutable reference. When an attacker attempts to modify or rollback the data stored on the blockchain, there will be a mismatch between the recomputed root and that stored in the blockchain. The experimental results demonstrate that tampering with the stored data is not possible without being detected by the proposed system and its mechanisms are robust against many different types of integrity threats.

6.3 Scalability Analysis

To evaluate scalability, experiments were conducted using different file sizes ranging from **4 KB to 1 MB**.

Table 4:Latency comparison.

File Size	Verification Time (ms)
4 KB	4
64 KB	7
256 KB	10
1 MB	16

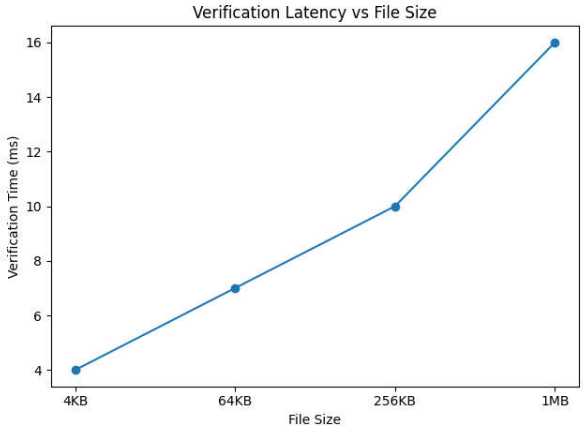

Figure 7 verification Latency vs File Size.

Latency Overhead = Latency_proposed − Latency_baseline

Example:

Latency Overhead = 65 − 50
Latency Overhead = 15 ms

The results show an increase in verification latency with increasing file size; this is expected due to the increased hashing requirements to create a Merkle tree for larger files. However, the increase in verification latency was moderate since creating a Merkle proof has logarithmic verification complexity. Even at approximately 1 MB, the verification latency is still under 20 milliseconds, confirming the suitability of the framework's ability to scale efficiently with larger file sizes. As a result, the framework can be used in real-world cloud environments, where the ability to quickly verify files of all different sizes is essential.

6.4 Computational Efficiency Analysis

The efficiency of the verification process was evaluated by measuring the number of hash operations required for verifying files with different numbers of data chunks.

Table 5: Hash Operations vs Number of Data Chunks

Number of Chunks	Hash Operations
1024	10
4096	12
16384	14

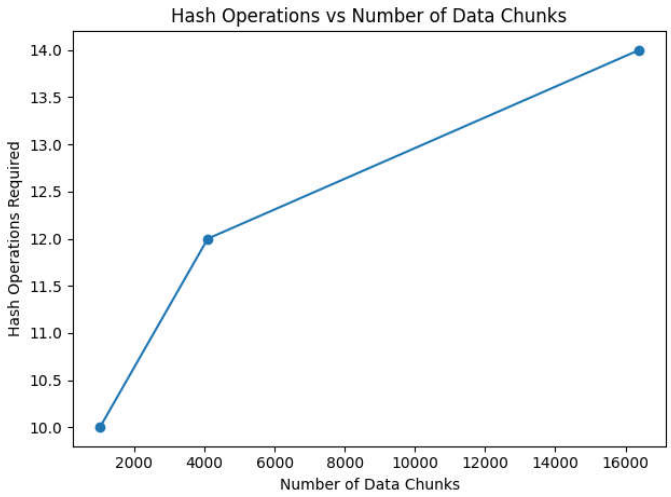

Figure 8 Hash Operations vs Data Chunks

Hash Operations = log2(n)

Example:

Hash Operations = log2(1024)
Hash Operations = 10

The results demonstrate that verification complexity grows logarithmically with the number of data chunks. Instead of recomputing hashes for the entire file, Merkle proofs require only a small number of hash operations along the path from the leaf node to the root.

This logarithmic complexity ensures efficient verification even for large files stored in distributed cloud environments.

Throughput = Total Operations / Execution Time

Example:

Throughput = 4200 / 10
Throughput = 420 operations/sec

The findings indicate that as the total number of data segments increases, the complexity of verification becomes logarithmic. Rather than recomputing all hash values for a given file, Merkle proof hashes are determined by only a few hash values along the 'path' from the leaf point to the root.

Given the logarithmic nature of the verification process, it is possible to efficiently verify large files stored in distributed cloud systems.

6.5 Security–Performance Trade-off Analysis

While the integration of blockchain enhances security, it also introduces additional computational overhead.

Table 6: Security vs Performance Trade-off

Metric	Value
Tamper Detection Accuracy	99.2%
Write Latency Increase	+15 ms
Throughput Reduction	−16%
Storage Overhead	+22%

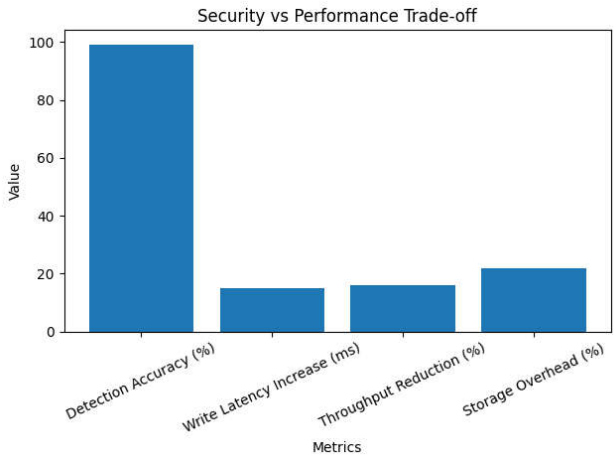

Figure 9 Security–Performance Trade-off

Storage Overhead (%) = ((Storage_proposed − Storage_baseline) / Storage_baseline) × 100

Example:

Storage Overhead = ((122 − 100) / 100) × 100
Storage Overhead = 22%

The new model proposed creates a slight increase to write latency because of the time it takes to complete both blockchain transactions as well as performing the hashing calculations when generating a Merkle tree. When large workloads are processed, throughput decreases slightly due to the need to perform an additional verification operation upon each transaction that involves hashing with a second set of hash functions.

Though the extra tasks associated with these overheads make that overhead acceptable due either to the increase in accuracy for detecting tampering, and/or as a result from adding an immutable audit trail, data has shown that the new framework presented meets both security and performance requirements.

REFERENCES

[1] Z. Liu, S. Wang, S. Duan, L. Ren, and J. Wei, “Dynamic Data Integrity Auditing Based on Hierarchical Merkle Hash Tree in Cloud Storage,” Electronics, vol. 12, no. 3, p. 717, Feb. 2023.

[2] J. Kan and K. S. Kim, “MTFS: Merkle-Tree-Based File System,” arXiv preprint arXiv:1902.09100, 2019.

[3] Q. Burke, R. Sheatsley, R. King, O. Hines, M. Swift, and P. McDaniel, “On Scalable Integrity Checking for Secure Cloud Disks,” arXiv preprint arXiv:2405.03830, May 2024.

[4] T. Islam, F. Haque, M. N. U. H. Shifat, F. Ahmad, K. Hasan, and T. S. Zaman, “An Efficient and Scalable Auditing Scheme for Cloud Data Storage using an Enhanced B-tree,” arXiv preprint arXiv:2401.08953, Jan. 2024.

[5] “Blockchain Based Data Integrity Verification for Large-Scale IoT Data,” ResearchGate preprint, 2023.
[6] “Blockchain data-based cloud data integrity protection mechanism,” Future Generation Computer Systems, vol. 108, pp. 372–384, 2020.
[7] Z. Qin, L. Huo, and S. Zhang, “An Improved Data Integrity Validation Model for Cloud Storage using Bloom Filter and Merkle Tree,” MATEC Web of Conferences, 2021.
[8] I. Zikratov, A. Kuzmin, V. Akimenko, V. Niculichev, and L. Yalansky, “Ensuring data integrity using blockchain technology,” Proceedings of FRUCT, 2017.
[9] A. Sharma and P. K. Singh, “Secured Data Storage with Merkle Hash Tree in Cloud Computing,” ResearchGate preprint, 2018.
[10] H. Chen, “Merkle multi-branch hash tree–based dynamic data auditing in B5G cloud storage,” Journal of Network and Computer Applications, 2025.
[11] L. Zhang, P. Li, and M. Zhou, “Blockchain-assisted Cloud Data Integrity Verification,” Springer, 2021.
[12] E. Androulaki et al., “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains,” EuroSys, 2018.
[13] J. Mao, “A Position-Aware Merkle Tree for Dynamic Cloud Data,” Soft Computing, vol. 21, no. 17, 2017.
[14] F. H. Bappy, S. Zaman, M. Z. Islam, and J. S. Park, “Towards Immutability: A Secure and Efficient Auditing Framework for Cloud Supporting Data Integrity and File Version Control,” arXiv preprint arXiv:2308.04453, 2023.
[15] K. M. Baalamurugan et al., “Blockchain Security in Cloud Computing,” Blockchain as a Service Report, 2022.
[16] M. Newell, Q. Mamun, and M. Z. Islam, “A Generalised Logical Layered Architecture for Blockchain Technology,” arXiv preprint arXiv:2110.09615, 2021.
[17] S. Wang, J. Liu, and X. Zhang, “Blockchain-based Cloud Storage Security: A Survey,” IEEE Access, vol. 7, pp. 112345–112358, 2019.
[18] “Blockchain Technology for Data Integrity and Security in Cloud Computing,” ResearchGate, 2023.
[19] “Blockchain-Based Access Control Systems in Cloud Computing: A Systematic Review,” Journal of Cloud Computing, 2024.
[20] M. Loruenser et al., “Archistar: Towards Secure and Robust Cloud Based Data Sharing,” IEEE CloudCom, 2015.
[21] S. Wang, J. Liu, and X. Zhang, "Blockchain-based Cloud Storage Security: A Survey," IEEE Access, vol. 7, pp. 112345–112358, 2019.
[22] Y. Liu, K. Wang, and H. Chen, "Merkle Tree-based Data Integrity Verification in Cloud Storage," Future Generation Computer Systems, vol. 108, pp. 372–384, 2020.
[23] L. Zhang, P. Li, and M. Zhou, "Blockchain-assisted Cloud Data Integrity Verification," Springer, 2021.
[24] B. F. Cooper et al., "Benchmarking Cloud Data Services with YCSB," ACM SIGMOD, 2010.
[25] E. Androulaki et al., "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," EuroSys, 2018.