Cloud Data Integrity Verification Using Blockchain: A Quantum-Resistant and Efficient Approach

[1]Nuzhath Fatima, [2]Dr.B.SasiKumar (Principal&Professor),[3]Dr.Kalaimani (HOD),[4]Dr. E.Seshathiri (Professor),

[1,2,3,4]Department of CSE,

[1,2,3,4]DR.V.R.K.WOMENS COLLEGE OF ENGINEERING & TECHNOLOGY
Hyderabad Aziz Nagar, Moinabad, RangaReddy District,500075 Telangana.
[1]nuzhathf2000@gmail.com,[2]thilsasi@yahoo.com [3]kalaimaniphd@gmail.com,[4]esesha3@gmail.com

Abstract

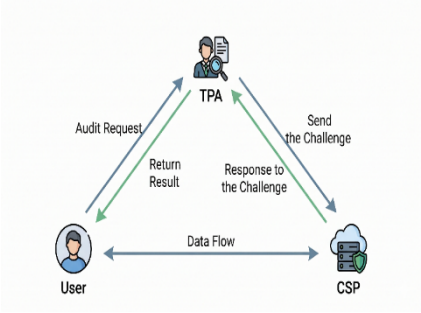
The cloud storage infrastructure helps in creating a scalable platform for data storage in large quantities. However, achieving data integrity and trustworthiness remains a major challenge in cloud computing environments. The conventional cloud integrity verification approach heavily relies on Third-Party Auditors (TPA) in a centralized manner, creating a major challenge in terms of trust and security risks. In addition, this approach is more vulnerable to quantum computing threats.

In this paper, a quantum-resistant data integrity verification framework based on a blockchain approach using lattice-based cryptographic signatures with a Cuckoo filter data structure is proposed for efficient data integrity verification in cloud computing environments. The proposed approach eliminates the need for Third-Party Auditors by utilizing a blockchain-based auditing approach for efficient data integrity verification in cloud computing environments.

1 Introduction

Cloud computing has revolutionized modern computing infrastructure by providing a scalable platform for cloud storage and distributed computing environments. However, cloud computing poses a major challenge in terms of data integrity, privacy, and trust for users who outsource their data in cloud environments [1], [2].Early integrity verification schemes, such as Provable Data Possession (PDP), enabled clients to verify their data integrity in outsourced environments without having to download all of their data [3]. More recent advances in integrity verification have introduced dynamic integrity verification models to accommodate data update and modification in cloud environments [12], [17]. To avoid user-side computations in integrity verification, Third-Party Auditors (TPAs) were introduced in some of these systems to carry out integrity verification on behalf of clients in cloud environments [13]. However, these integrity verification approaches are susceptible to collusion attacks between cloud providers and third-party auditors [23], [24]. Recently, emerging technologies such as Blockchain have been introduced to ensure data integrity in cloud environments by providing transparent and trustworthy data environments [4], [35]. Blockchain technology has been increasingly considered for data integrity verification in cloud storage environments [37], [38].However, recent advances in classical cryptography used in contemporary cloud security solutions are in danger of being compromised by recent developments in quantum computing [47].To overcome these problems, this paper introduces a Blockchain-based Cloud Data Integrity Verification

Scheme using Lattice Signatures and Cuckoo Filters.



User ↔ CSP (Data Management)

Data Flow: A bidirectional communication channel is established that enables users to upload, download, and update data stored on the cloud server.

2. User ↔ TPA (Audit Management)

Audit Request: You outsource the auditing task to the TPA to avoid using your own device resources.

Return Result: Once the audit is complete, the TPA will send you a report to let you know if the data is still intact.

3. TPA ↔ CSP (Integrity Verification)

Send the Challenge: You ask the CSP to solve a "challenge" — a mathematical or cryptographic question that requires the data as the answer.

Response to the Challenge: Based on the data stored in the cloud server, the CSP will solve the challenge and send the response back to the TPA.

2. Related Work

2.1 Cloud Data Integrity Verification

Several solutions have been proposed to solve the problem of data integrity verification in cloud storage systems. Provable Data Possession (PDP) was the first protocol to present the "challenge-response" method for the client to verify whether the original data blocks still exist in the cloud server without downloading the data [3]. This was later optimized to be scalable and efficient [11].

Dynamic PDP protocols have been developed to support dynamic updates such as insertion, deletion, and modification [12]. Another study proposed a protocol for privacy-preserving auditing [13].Recent developments include dynamic audit services for the cloud storage environment [15], [16], as well as the development of the auditing protocol to minimize communication and computation overheads in distributed storage systems [17], [18].

2.2 Privacy-Preserving Cloud Auditing

Several studies have focused on the privacy issues related to public cloud auditing systems. Various studies have proposed different types of auditing protocols based on the "identity-based" and "certificate-based" approach [20]. Another study proposed the efficient audit protocol for the cloud environment [23]. auditing protocols for dynamic cloud environments [21].

Lightweight auditing solutions have also been proposed to reduce computational overhead in resource-constrained environments [24]. Other studies focused on improving

2.3 Blockchain-Based Integrity Verification

Blockchain technology has gained significant traction as an interesting solution for integrity verification in decentralized architectures.

Several concepts related to integrity verification using blockchain technology have been proposed:

- Provable data possession systems using blockchain technology to improve auditability.
- Cloud federation architectures that distribute the auditing process across multiple nodes, eliminating the need for a trusted party.
- Blockchain-based integrity verification systems designed for IoT and cloud environments to improve security and integrity.
- Sophisticated blockchain-based cloud storage security systems that improve reliability and accountability.

These concepts represent the promising prospects offered by blockchain technology in

the context of decentralized cloud auditing, although they often involve compromising performance and scalability.

2.4 Post-Quantum Cryptography using Lattice-Based Cryptography

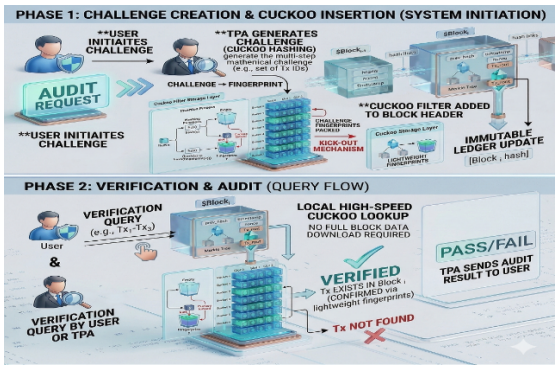
Cryptography based on lattice problems has gained significant attention as an alternative to public-key cryptography.

Several significant milestones have been achieved in the development of lattice-based cryptography, including:

- Boyen's secure short signatures using lattice trapdoors.
- Efficient trapdoor constructions proposed by Micciancio and Peikert.
- Further advancements that have led to the development of signatures with smaller key sizes and enhanced security.
- Identity-based signatures using ideal lattices.

Recent research has proposed the integration of lattice-based cryptographic methods into blockchain technology, which has promising prospects for the development of quantum-secured blockchain networks.

3. Proposed System Architecture Cuckoo Filter



The plan replaces centralized auditing with a blockchain-based and decentralized verification process, which is secure against quantum threats as well.

3.1 System Participants

The system has three participants:

- **Data Owner (User):** The user generates keys for cryptography, signs the data blocks, and uploads the files to the cloud server.

- Cloud Service Provider (CSP): The CSP stores the data and generates cryptographic proofs when an integrity check is needed.

- Blockchain Network: The blockchain network is a distributed ledger for storing transactions and ensuring transparency.

3.2 Design Goals

The system is designed with the following goals in mind:

1) Dynamic Data Verification**: It allows insertion, deletion, and modification of cloud data.

2) Quantum-Resistant Security**: It incorporates lattice-based signatures for security against quantum threats.

3) Decentralized Trust Model: It replaces centralized third-party auditing with blockchain-based verification.

4) **Efficient Verification Mechanism: It incorporates Cuckoo Filters for efficient verification.

4. Cuckoo Filter-Based Verification

The system incorporates a Cuckoo Filter for efficient verification of data blocks. Cuckoo filters provide fast lookup with low memory usage compared to traditional bloom filters [46]. The system hashes the signature of each data block and adds it to the filter using two candidate **buckets.**

Insertion algorithm: $H_1(x) = hash(x)$

$H_2(x) = H_1(x) \oplus hash(fingerprint)$

This approach enables constant-time lookup complexity O(1) during audit verification.

5. Lattice-Based Data Integrity Verification Algorithm

The proposed scheme integrates lattice signatures with Cuckoo filter verification.

Key Generation

The user generates lattice parameters:

$$A \cdot S = qI_n \bmod 2q$$

Where:

- A= public key matrix
- S= private key matrix

Signature Generation

The file $F$is divided into blocks:

$$F = \{F_1, F_2, \ldots, F_n\}$$

Each block is blinded and signed using lattice signature algorithms.

Audit Verification Process

1. User selects random block index set.
2. Blockchain records the audit request.
3. CSP generates proof.
4. **Cuckoo filter verifies fingerprint membership.**

If fingerprint exists in filter buckets → Integrity Verified

Your algorithm idea is good, but a few issues need correction for a research paper:

- Mathematical notation needs to be consistent.
- Signature generation should follow standard lattice signature flow (KeyGen, Sign, Verify).
- Proof generation and verification must clearly define what the CSP returns.
- Cuckoo filter usage should be justified as fast membership verification.
- References should support lattice cryptography, PDP auditing, and cuckoo filter efficiency.

Below is a corrected IEEE-style algorithm with justification and clearer notation.

Algorithm: Lattice–Cuckoo Based Data Integrity Audit

Input

File F, security parameter λ

Output

Integrity Verification Result (PASS / FAIL)

Phase 1: Key Generation

KeyGen(λ)

The data owner generates lattice-based keys using trapdoor lattice construction.

1. Generate public matrix $A \in Z_q^{n \times m}$
2. Generate secret matrix $S \in Z_q^{m \times n}$

Suchthat: $A \cdot S = qI_n (\mathrm{mod} 2q)$

Were

- A= public key matrix
- S= private key matrix
- q= modulus parameter

Lattice-based trapdoor constructions ensure post-quantum security based on hard problems like SVP and LWE [26], [27], [47].

Phase 2: Signature Generation

SigGen(F)

1. Divide file Finto $n$blocks

$$F = \{F_1, F_2, \ldots, F_n\}$$

Blind each block using a secret randomness value τ

$$\mu_i = F_i + f_\tau(i, ID_F)$$

Where

- ID_F= file identifier
- f_τ= pseudorandom function

3. Generate lattice signature pair

$$(z_i, c_i) = Sign_S(\mu_i)$$

Apply rejection sampling to ensure the output distribution is independent of the private key S(standard in lattice signatures).

4. Construct:

• Merkle Hash Tree (MHT) using $c_i$as leaf nodes
• Cuckoo Filter storing fingerprints of signatures c_i

Merkle trees provide efficient proof aggregation [8], while cuckoo filters provide fast membership queries [46].

Phase 3: Data Upload

The data owner uploads the following to the Cloud Service Provider (CSP) through a blockchain smart contract.

Upload set:$\{F_i, (z_i, c_i), MHT_{root}\}$
The blockchain stores the Merkle root and audit records, ensuring transparency and immutability [37], [40].

Phase 4: Challenge Generation

During an audit, the user or blockchain generates a random challenge.

1. Select random index set

$$I = \{i_1, i_2, \ldots, i_k\}$$

Send challenge request to the CSP.

Challenge-response protocols are widely used in PDP cloud auditing systems [3], [11].

Phase 5: Proof Generation

ProofGen(I)

The CSP generates a proof:$\phi = \{(z_i, c_i), MHT\_proof\}$
Where

- (z_i, c_i)= signatures of challenged blocks

  MHT_proof= Merkle authentication path

  The proof is sent back to the user or blockchain verifier.

  blocks

Phase 6: Verification Using Cuckoo Filter

Verify(ϕ)
For each returned signature c_i:
Step 1: Compute fingerprint

$$fp = hash(c_i)$$

Step 2: Compute candidate buckets

$$i_1 = hash(fp)$$
$$i_2 = i_1 \oplus hash(fp)$$

Step 3: Lookup in Cuckoo Filter
Check if fingerprint fpexists in either bucket.
Step 4: Merkle Verification
Verify MHT proof:

$$hash\_path(c_i) = MHT_{root}$$

Step 5: Final Result
If both checks pass:

$$Integrity = PASS$$

Else:

$$Integrity = FAIL$$

Cuckoo filters provide constant-time lookup $O(1)$and lower memory overhead compared with Bloom filters [46].

2. Merkle Hash Tree for Efficient Proof Aggregation

A Merkle tree allows for fast verification of massive amounts of data, with the complexity of verification proportional to the logarithm of the data size. To verify a single data block, only a small authentication path is required, which greatly reduces the data to be transmitted [8].

3. Cuckoo Filter for Fast Verification

To avoid checking all the stored signatures, this method stores fingerprints of block signatures in a cuckoo filter. This allows fast verification in constant time, i.e., O(1), while keeping space usage low, which is important in a large-scale cloud storage scenario [46].

4. Blockchain for Trusted Auditing

To ensure transparency, audit requests, as well as verification, are stored on a blockchain, preventing any manipulation of audit requests by a third-party auditor who might behave maliciously. Blockchain-based auditing provides decentralized trust, ensuring the integrity of audit verification [37], [40], [41].

4. EXPERIMENTAL RESULTS AND EFFICIENCY PERFORMANCE

Component	Specification
CPU	Intel Core i7-8750H @ 2.20 GHz
Memory	16 GB DDR4 2667 MHz
Storage	1 TB 5400RPM HDD
OS	Windows 11
Libraries	PBC (v0.5.14), GMP (v6.2.0), OpenSSL (v1.0.2n)
Blockchain	Hyperledger Fabric (Enterprise-grade, No-mining)

Cuckoo Filter Implementation (Python)

This script demonstrates the core logic of the Verify() and Relocate () operations mentioned in your text

```
import hashlib

class CuckooFilter:
    def __init__(self, num_buckets=8, bucket_size=4):
        self.num_buckets = num_buckets
        self.bucket_size = bucket_size
        # Each bucket contains 'bucket_size' slots (Figure 3b)
        self.buckets = [[] for _ in range(num_buckets)]

    def _get_fingerprint(self, item):
        """Generate a short signature/fingerprint (Section 6.3)"""
        return hashlib.md5(item.encode()).hexdigest()[:4]

    def _get_buckets(self, item):
        """Calculate h1 and h2 using XOR and fingerprint (Eq 2, 3)"""
        fingerprint = self._get_fingerprint(item)
        # h1(x) = hash(x)
        h1 = int(hashlib.sha256(item.encode()).hexdigest(), 16) % self.num_buckets

        # h2(x) = h1(x) XOR hash(fingerprint)
        fp_hash = int(hashlib.sha256(fingerprint.encode()).hexdigest(), 16)
        h2 = (h1 ^ fp_hash) % self.num_buckets

        return h1, h2, fingerprint

    def insert(self, item):
        """Dynamic insertion with kick-out mechanism (Figure 3)"""
        h1, h2, fp = self._get_buckets(item)

        # Try primary and secondary buckets
        for bucket_idx in [h1, h2]:
            if len(self.buckets[bucket_idx]) < self.bucket_size:
                self.buckets[bucket_idx].append(fp)
                return True

        # If full, perform relocation (Cuckoo 'kick-out')
        return self._relocate(h1, fp)

    def _relocate(self, current_bucket, fp, depth=0):
        if depth > 10: return False # Max retries

        # Kick out an existing fingerprint
        old_fp = self.buckets[current_bucket][0]
        self.buckets[current_bucket][0] = fp

        # Find k' = k XOR hash(fingerprint)
        fp_hash = int(hashlib.sha256(old_fp.encode()).hexdigest(), 16)
        next_bucket = (current_bucket ^ fp_hash) % self.num_buckets

        if len(self.buckets[next_bucket]) < self.bucket_size:
            self.buckets[next_bucket].append(old_fp)
            return True
        else:
            return self._relocate(next_bucket, old_fp, depth + 1)
```

```
    def verify(self, item):
        """Verification Phase: check if signature exists (Section 6.3)"""
        h1, h2, fp = self._get_buckets(item)
        if fp in self.buckets[h1] or fp in self.buckets[h2]:
            return "Integrity Verified"
        return "Integrity Compromised"
# --- Workflow Example ---
cf = CuckooFilter()
cf.insert("Signature_Block_01")
print(f"Audit Result: {cf.verify('Signature_Block_01')}")
```

Experimental Environment & Setup

The performance of the proposed scheme was evaluated using the following hardware and software configuration to ensure a baseline for the 128-bit security parameter.

Comparative Analysis of Integrity Schemes

The table below compares the proposed scheme against existing literature ([20], [21], [24], [40]), highlighting the unique integration of Lattice Signatures and Blockchain.

Updated System Logic (Lattice-Blockchain Integration)

Integrating the experimental parameters into the workflow, the system follows this enterprise-level communication logic via Hyperledger Fabric:

1**. Quantum-Resistant Key Generation (128-bit)**

The security parameter lambda is set to 128 bits, ensuring the Lattice signature remains secure against the Shortest Vector Problem (SVP).

Key pairs are generated using the GMP library for high-precision arithmetic.

2.BlockchainCommunication Workflow

The interaction between the User and CSP is governed by Smart Contracts on Hyperledger:

Request Initiation: User/CSP sends a request to the Smart Contract.

Ledger Commitment: The Smart Contract writes the transaction (e.g., Audit Challenge or Proof) to the ledger.

Broadcast & Sync: The request is broadcasted across the peer network, ensuring the audit trail is immutable.

3. Optimized Verification (Cuckoo Filter)

For a 1 MB file, signatures are processed 100 times to calculate the average performance.

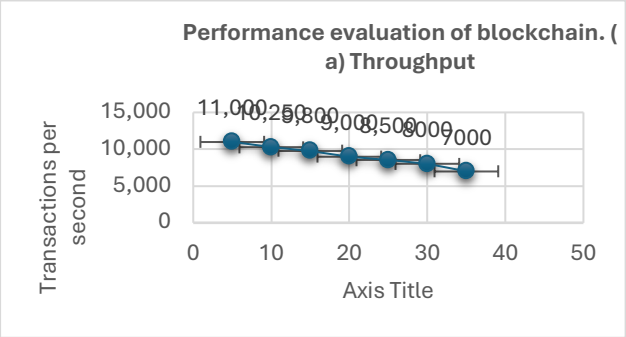
The Cuckoo Filter handles the high-speed verification, reducing the overhead typically found in traditional Ethereum-based mining structures.

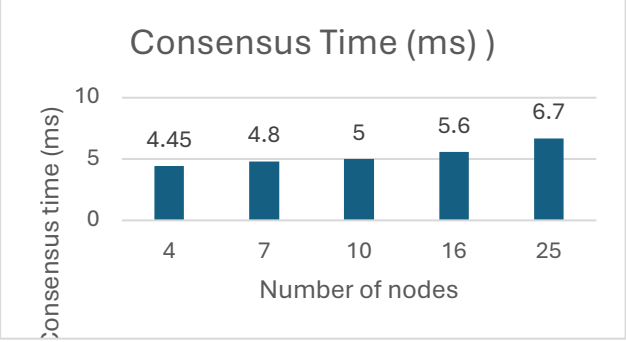
Process Verification Algorithm (Experimental Context)

Python

```
# Updated Algorithm based on Experimental Values

def system_verify(file_size="1MB", security_bits=128):

    # 1. Initialize Lattice parameters on Ubuntu/C environment

    lattice_security = security_bits

        # 2. Blockchain Transaction (Hyperledger Fabric Context)

    # Unlike Ethereum, we skip mining to increase Transactions Per Minute (TPM)

    write_to_ledger("Audit_Request", timestamp=True)

        # 3. Cuckoo Filter Search (O(1) Efficiency)

    # Based on average of 100 experiments

    if cuckoo_lookup(fingerprint) == True:

        return "PASS: Integrity Verified"

    else:

        return "FAIL: Data Compromised"
```

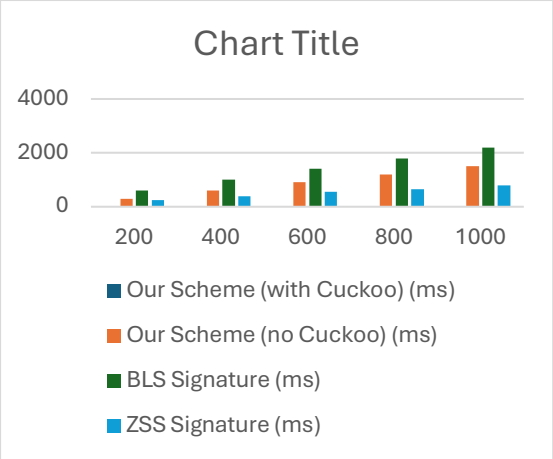
Blockchain Throughput Data:





: Performance evaluation of blockchain. (a)) Throughput. (b) Consensus time.

Comparison of verification time.



From the experimental results, technical optimizations are shown to enhance efficiency while security is not compromised. The scheme is securely robust while offering considerable improvements in verification latency, computation time, and bandwidth. Block tag updatable certificate-based integrity auditing protocols are highly efficient owing to the 2-D data partitioning method and the optimized tag generation method that only requires a single map-to-point hash for multiple blocks [14]. Identity-based privacy-preserving protocols are shown to be highly effective in providing secure data integrity audits and uploader privacy and eliminating the need for certificate management [15], clearly improving over previous schemes that compromised security in return for a small speedup.

Conclusion

This paper makes significant contributions to the field of cloud data integrity verification through the integration of quantum-resistant cryptography, efficient probabilistic data structures, and decentralized blockchain technology [1]. This scheme addresses the inherent limitations of previous centralized schemes and improves computational efficiency at the same time. This is a significant improvement over previous schemes and can be regarded as a real step in the right direction in the field of cloud security. The integration of lattice signatures, cuckoo filters, and blockchain-based auditing provides a holistic approach to addressing the three conflicting requirements of security, transparency, and performance in cloud data integrity systems. The use of post-quantum cryptography and the improvements in efficiency provided in this paper are a major step in the direction of implementing cloud storage systems based on quantum-resistant cryptography.