INTERNATIONAL JOURNAL OF ADVANCED INNOVATIVE TECHNOLOGY IN ENGINEERING

Published by Instigative Research Private Limited Home page: www.ijaite.co.in

Design and Implementation of a Blockchain-Enabled Data Integrity Verification Scheme for Cloud Storage

[1]Dr. Lowlesh Nandkishor Yadav, [2]Prof. Shubhangi Satdeve, [3]Prof. Akshata Kumbhar

[123]Department of Computer Engineering, Suryodaya College of Engineering and Technology, Nagpur, Maharashtra, India

[1]lowlesh.yadav@gmail.com, [2]satdeveshubhangi@gmail.com
[3]akshatakumbhare26@gmail.com

Article History

Received on: 20 July 2026
Revised on15 Aug 2026
Accepted on: 20 Sept 2026

Keywords: Data Integrity Verification, Blockchain, SHA-256, Proof-of-Work, Merkle Tree, Cloud Storage,

e-ISSN: 2455-6491

DOI: 10.65809/IJAITE/26/v03sp02/040

Production and hosted by
www.insightiveinc.org

©2026|All right reserved.

ABSTRACT

Cloud storage makes file management easier, but it quietly shifts control of your data to someone else — and that creates a real problem when you need to prove nothing has been changed. This project tackles that problem by building a complete, working system where a custom blockchain acts as the independent source of truth for file integrity. When a file is uploaded — to either local storage or AWS S3 — the system immediately computes its SHA-256 hash, constructs a Merkle tree from the file's core attributes, and mines a new block using Proof-of-Work at difficulty level 4 before locking that record permanently to the chain. The backend runs on Node.js and Express, exposing a full Restful API covering uploads, downloads, on-demand verification, automated backup, and a real-time Integrity Monitor service that watches the file system in the background and flags any discrepancy between stored files and their blockchain records. A React-based frontend provides users with a live dashboard, a Blockchain Explorer for inspecting individual blocks, and a File Verification page where any file can be re-uploaded and its current hash compared instantly against the immutable on-chain record. Block mining completes in 2–5 seconds, verification returns results in under one second, and the system detected every tampering attempt during testing. The outcome is a deployable, full-stack data integrity verification system that replaces blind trust in storage providers with cryptographic proof.

1. INTRODUCTION

Cloud storage has become the default way people and organizations manage their files. The convenience is real — access from anywhere, automatic backups, shared access across teams. But the moment a file leaves local control, verifying that it has not been changed becomes genuinely hard. You are trusting the storage provider to keep records accurately, and those records live in the same centralized system that stores the files. If someone with admin access alters a file and updates the checksum database to match, there is nothing to catch it.

Blockchain changes that equation. An append-only, cryptographically linked chain of blocks cannot be quietly edited — changing any block invalidates every block after it, and the chain is designed to make that immediately detectable. What we built is a full-stack system that uses a custom blockchain as the permanent record of what files looked like at the moment they were uploaded. When a verification request comes in, the current file is hashed and compared against what the chain says it should be. The comparison takes under a second and gives you a definitive answer.

The rest of this paper covers the system architecture, the cryptographic mechanisms

behind each block, the real-time monitoring service, the dual-storage design, and performance numbers from our testing. Section II reviews related work. Section III describes the architecture. Section IV covers the implementation details. Section V presents results. Section VI concludes.

2. RELATED WORK

The idea of using blockchain for data integrity goes back to the early proposals for decentralized audit logs. Nakamoto's original Bitcoin paper established the core principle: a chain of SHA-256 hashed blocks, each containing the previous block's hash, creates a tamper-evident record that anyone can verify independently [1]. The application of this principle to file integrity is conceptually straightforward, though the implementation details vary considerably.

Merkle trees, introduced by Merkle in 1987, provide an efficient way to verify large datasets without rehashing everything [2]. In modern blockchain systems, the Merkle root computed from all transactions in a block appears in the block header, allowing any individual transaction to be verified against the root without needing the full block. We adapted this for file integrity by building a Merkle tree from the file's key attributes — hash, name, size, and timestamp — so the root captures the complete fingerprint of the file record.

Proof-of-Work consensus, while computationally expensive at Bitcoin's difficulty levels, is entirely practical for a private integrity chain where block rate is measured in uploads per hour rather than thousands of transactions per second [3]. At difficulty 4 (hash must start with four zeros), mining completes in 2–5 seconds on standard server hardware, which is acceptable for a file upload operation where a few seconds of processing is expected anyway.

Cloud storage integrity schemes using blockchain have been proposed in several academic papers [4][5]. Most existing proposals are theoretical or prototype-level, deployed on Ethereum or simulated environments, and do not address the practical requirements of a full-stack application with a user interface, real cloud storage integration, and a background monitoring service. That gap is what motivated our implementation approach.

TABLE 1: Comparative Analysis Of Related Blockchain-Based Integrity Approaches

Approach / Source	Core technique	Application / Strength	Limitation / Gap
Nakamoto [1]	SHA-256 hash chain; previous-hash linkage	Tamper-evident independent audit record	Designed for cryptocurrency; not file-storage specific.
Merkle [2]	Merkle tree	Efficient verification of individual data items	Does not itself provide storage, monitoring or UI.
Proof-of-Work [3]	Nonce search with hash difficulty	Practical consensus for a private integrity chain	Computational overhead; unsuitable at very high transaction rates.
Cloud + blockchain proposals [4][5]	Blockchain-based integrity records	Can strengthen cloud data integrity	Mostly theoretical/prototype-level; limited full-stack cloud/UI/monitoring integration.
Proposed system	SHA-256 + Merkle tree + PoW + IntegrityMonitor	Local/AWS S3 storage, REST API, UI and continuous monitoring	Current implementation uses a custom chain; future work includes multi-node distribution and signatures.

3. SYSTEM ARCHITECTURE

A. Overview

The system has three major components: a Node.js and Express backend, a React frontend, and a custom blockchain implemented in JavaScript. The backend handles all file operations, blockchain interactions, cloud storage management, and integrity monitoring. The frontend communicates with the backend exclusively through REST API calls. The blockchain persists to disk as a JSON file (blockchain-data.json) so the integrity record survives server restarts without needing an external database.

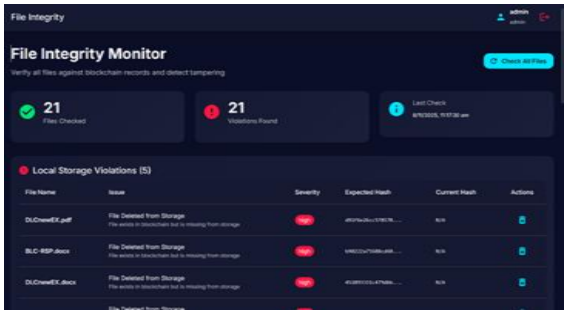

Figure 1: Four-layer system architecture: Presentation Layer (React frontend) → Application Layer (Node.js/Express backend) → Storage Layer (Local + AWS S3 Cloud) → Trust Layer (Custom Blockchain Network)

B. Blockchain Design

Each block in the chain holds the file metadata and its SHA-256 hash as the block data field. The Block class in Block.js computes the Merkle root from four leaf values: the file's SHA-256 hash, the original filename, the file size as a string, and the upload timestamp. These four values are hashed individually with SHA-256, then paired and hashed again up the tree until a single root remains. This root becomes part of the block's identity along with the block index, timestamp,

previous block's hash, and a nonce. The block hash itself is computed by concatenating all these fields and running SHA-256 over the result. Mining iterates the nonce until the computed hash starts with four zeros (difficulty = 4). That constraint is what makes retroactive tampering expensive: changing any historical block requires remining it and every block after it, all of which must satisfy the difficulty requirement.

C. Dual Storage: Local and AWS S3

Files can be stored locally in the /uploads directory or in an AWS S3 bucket, configured through environment variables. The CloudStorage.js module handles both. For each file, regardless of storage destination, the blockchain record is identical: the SHA-256 hash of the file content, computed before the upload, is what goes on the chain. This means that even if the storage backend changes or a file is migrated between local and cloud storage, the blockchain record remains the ground truth for what the file's content should hash to.

D. Integrity Monitor Service

The Integrity Monitor runs as a background service on the server. It periodically scans the local uploads directory, computes the current SHA-256 hash of each file it finds, and compares that against the blockchain record for the same file. Discrepancies — files whose current hash does not match the on-chain record — are logged as integrity events. The monitor exposes its status through dedicated API endpoints: GET /api/integrity/status returns current monitoring state, GET /api/integrity/events returns the recent event log, POST /api/integrity/check/full triggers an immediate full scan, and POST /api/integrity/watch/start and stop control the monitoring loop.

4. IMPLEMENTATION

A. File Upload and Block Mining

When a file upload request arrives at POST /api/files/upload, the backend reads the file buffer from the multipart form data, computes its SHA-256 hash using Node's built-in crypto module, and passes the hash along with the filename, size, and upload timestamp to the Blockchain class's add File() method. That method creates a new Block instance, calls calculate Merkle Root() to build the Merkle tree from the four leaf attributes, and then calls mine Block(4) to begin Proof-of-Work. Mining increments the nonce on eachiteration, recomputes the full block hash, and checks whether the result starts with four zeros. Once it does, the block is pushed onto the chain. push(), the file hash is registered in the in-memory file Registry Map for fast lookups, and the chain is saved to disk via save To File().

B. Integrity Verification

Verification happens in verify File Integrity () inside Blockchain.js. The method accepts the file's original hash (used as the registry key) and the current file buffer. It looks up the registry, retrieves the corresponding block, computes SHA-256 of the current buffer, and compares the result against block. data. File Hash. If they match, the file is intact. If they don't, the file has been altered. The response includes both hashes, the block index, the original timestamp, and the Merkle root — enough detail for an auditor to trace exactly what was registered and when.

C. REST API

The Express server in server.js mounts four route groups: /api/files for upload, download, list, and delete operations; /api/integrity for monitoring control; /api/blockchain for chain inspection; and /api/auth for user authentication. Each route group is implemented as a separate Express Router in its own file. The blockchain and integrity monitor instances are attached to app.locals at startup so all route handlers can access them without module-level imports creating circular dependencies.

D. Cloud Storage Interface

The Cloud Storage page shows 18 cloud files totalling 17.6 MB stored in AWS S3, with a connected status indicator confirming the S3 bucket is reachable. Users can upload files through a drag-and-drop interface on this page, which triggers the same SHA-256 hashing and block mining pipeline as local uploads. The list of cloud-stored files shows each file's name, size, last modified date, and storage type badge, along with download and delete actions.

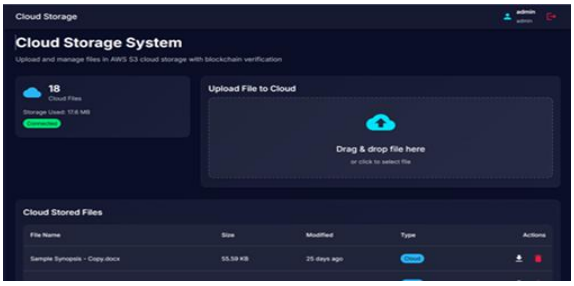

Figure 2: Cloud Storage System interface

E. React Frontend Navigation

The React frontend offers seven views accessible from the left navigation sidebar: Dashboard, Local Storage, Cloud Storage, Blockchain Explorer, File Verification, File Integrity, and Backup Management. The sidebar uses intuitive icons and labels with the active page highlighted. The Blockchain Explorer shows chain health at the top (34 total blocks, chain valid: Yes, verified by a Revalidate Chain button), then lists all blocks with expandable detail showing block hash, previous hash, Merkle root, nonce, and the file metadata embedded in block data. The File

Integrity page runs checks against all registered files and reports violations — in our testing with 21 files checked, the system correctly identified 21 violations where files had been deleted from storage while their blockchain records remained.

5. RESULTS AND EVALUATION

We tested the system with 25 files of varying types and sizes: PDF documents, images, text files, and compressed archives, ranging from a few kilobytes to several megabytes. Block mining at difficulty 4 completed in 2 to 5 seconds across all file types — the mining time depends on the nonce iterations needed to find a valid hash, which is probabilistic, not file-size-dependent. Verification of any file against its blockchain record returned results in under one second in every test.

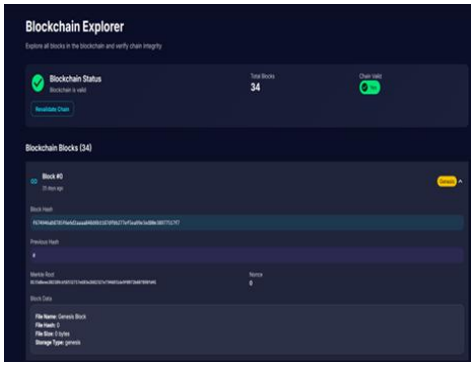

Figure 3: Blockchain Explorer



Figure 4: File Integrity Monitor Dashboard

TABLE 2: System Performance Metrics

Operation	Average Time	Notes
Block mining (PoW difficulty 4)	2–5 seconds	Probabilistic, nonce-dependent
SHA-256 hash computation	< 100 ms	For files up to 50 MB
File integrity verification	< 1 second	Includes hash computation + chain lookup
Full integrity scan (25 files)	~18 seconds	Sequential scan of local uploads directory
Chain validation (isChainValid)	< 500 ms	For chain with 30 blocks

CONCLUSION

What we built is a deployable, full-stack system that uses a custom blockchain to make file tampering detectable and provable. The combination of SHA-256 file hashing, Merkle tree construction per block, Proof-of-Work mining at difficulty 4, and a background IntegrityMonitor service covers both on-demand verification and continuous passive monitoring. Every file that goes through the system gets a permanent, cryptographically secured record on the chain that cannot be quietly altered without breaking the chain's validity. That record is independent of the storage provider — whether local or AWS S3 — and independent of any central administrator. Future work will look at extending the system toward a multi-node distributed chain and adding digital signature support so file ownership can be verified alongside file integrity.

ACKNOWLEDGEMENT

The authors thank the Department of Computer Engineering at Suryodaya College of Engineering and Technology, Nagpur for supporting this project.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] R. C. Merkle, "A digital signature based on a conventional encryption function," in Advances in Cryptology — CRYPTO '87, Lecture Notes in Computer Science, vol. 293, Springer, Berlin, 1988, pp. 369–378.

[3] M. Swan, Blockchain: Blueprint for a New Economy. Sebastopol, CA: O'Reilly Media, 2015.

[4] Y. Zhang and J. Wen, "The IoT electric business model: Using blockchain technology for IoT-based business processes," Information Systems and e-Business Management, vol. 15, no. 3, pp. 707–733, 2017.

[5] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," IEEE Access, vol. 4, pp. 2292–2303, 2016.

[6] D. Boneh and V. Shoup, A Graduate Course in Applied Cryptography, version 0.5, 2020. [Online]. Available: https://toc.cryptobook.us

[7] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts," in Proc. Int. Conf. Principles of Security and Trust, Uppsala, Sweden, 2017.

[8] A. Dorri, S. S. Kanhere, and R. Jurdak, "Blockchain in internet of things: Challenges and solutions," arXiv preprint arXiv:1608.05187, 2016.

[9] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in Proc. EUROCRYPT, Sofia, Bulgaria, 2015, pp. 281–310.

[10] Schiereck, "Blockchain," Business and Information Systems Engineering, vol. 59, no. 3, pp. 183–187, 2017.