Blockchain-Based Ransomware-Proof Backup System

Kandibanda Lohith, Lakshmi Priya P., Yaswanth Kancharla, Kavitha C.R.*
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
*Corresponding Author: cr_kavitha@blr.amrita.edu, ORCID: 0000-0003-1474-2243

***Abstract*—The increasing frequency and sophistication of ransomware attacks have exposed critical weaknesses in traditional backup systems, which are often susceptible to tampering, deletion, or encryption. To address these challenges, this paper presents a blockchain-based ransomware-proof backup system that integrates local AES-256 encryption, decentralized file storage via the InterPlanetary File System (IPFS), and hash-based integrity verification using Ethereum smart contracts. In the proposed system, user files are first encrypted using AES before being uploaded to IPFS, ensuring data confidentiality. A cryptographic hash of the encrypted file is then immutably recorded on the blockchain. During restoration, the system retrieves the file from IPFS, decrypts it using the user-provided key, and recomputes its hash to verify authenticity against the on-chain record. This design ensures that any modification of the stored file is reliably detected before restoration. Unlike conventional backup solutions, the system provides verifiable data integrity without relying on centralized servers or external trust. The use of blockchain ensures tamper-evidence, while IPFS guarantees high availability of encrypted content. This approach enhances resilience against ransomware attacks by enabling secure, decentralized, and auditable file backups and restores. The system is fully implemented using Flask, Web3.py, and smart contracts, and demonstrates a practical pathway toward trustless, ransomware-resistant data recovery.**

***Index Terms*—Blockchain, Ransomware, IPFS, Smart Contracts, Data Integrity, Cybersecurity, AES Encryption, Decentralized Backup**

I. Introduction

The exponential rise in ransomware attacks over the past decade has posed a critical threat to the confidentiality, availability, and integrity of data across sectors. These attacks, which typically involve the encryption of sensitive files followed by extortion demands, have evolved to also compromise backup infrastructures, thereby rendering recovery infeasible. Traditional backup methods—including on-premises file servers and cloud-based storage—remain vulnerable, as ransomware can locate and encrypt backup files or delete them entirely, undermining their intended purpose.

To address these escalating challenges, researchers have turned to blockchain-integrated security frameworks that leverage decentralization, immutability, and distributed consensus to enhance data protection. Systems such as BSFR-SH [1], [6] and RBEF [3] demonstrated how blockchain can be used to secure electronic health records against tampering. Other works have introduced access-controlled storage solutions [5], [9], blockchain-audited deduplication techniques [5], and intelligent backup strategies using machine learning for anomaly detection [10]. In addition, Delgado-Mohatar et al. [2] examined ransomware adaptation in blockchain environments, while Sokolov [7] analyzed blockchain congestion as a side effect of ransomware activity. Studies focusing on finance and sensitive domains further underscore the need for resilient infrastructure [4]. Additional insights have been gained from recent advances in ransomware detection and malware concealment strategies. Chandran et al. [11] presented AI-driven obfuscation detection techniques, while Achuthan et al. [12] reviewed cybersecurity trends combining federated learning and blockchain. Gurram et al. [13] explored the use of neural network models as covert carriers of malware, highlighting new attack surfaces. Vinayakumar et al. [14] demonstrated the utility of deep learning models in ransomware triage using social media data.

Despite these promising developments, most existing systems fall short in one or more key areas: they are often domain-specific (e.g., healthcare or finance), rely on semi-centralized enforcement mechanisms, or lack a verifiable, tamper-evident restoration process. Moreover, the absence of end-to-end validation from backup to restore limits the practical use of these systems in real-world ransomware resilience.

Table I summarizes selected blockchain-based ransomware mitigation systems and highlights their limitations in terms of decentralized storage, restore verifiability, and general-purpose applicability.

In this paper, we propose a ransomware-proof backup system that combines symmetric encryption, decentralized content-addressable storage, and tamper-evident blockchain validation. The system secures data confidentiality, distributes encrypted backups via IPFS, and uses smart contracts to anchor cryptographic hashes, enabling trusted and verifiable restoration.

Unlike prior approaches, our system introduces a general-purpose, trustless, and auditable restoration mechanism that operates without centralized oversight or static whitelists. It offers a resilient, scalable, and decentralized solution against ransomware threats, suitable for both organizational and individual deployment. The prototype is fully implemented using Flask, IPFS CLI, Web3.py, and Solidity-based smart contracts.

TABLE I
COMPARISON OF KEY BLOCKCHAIN-BASED RANSOMWARE MITIGATION SYSTEMS

Study	**Domain**	**Storage Type**	**Restore Verifiability**
[1], [6]	Healthcare	Blockchain + Cloud	No
[3]	Digital Health	Blockchain + Cloud	No
[5]	EHR Storage	Blockchain Audit	No
[9]	Access Drives	SSD + Ruleset	Partial
[10]	General	Local + Real-Time Backup	No
Ours	General Purpose	AES + IPFS + Ethereum	**Yes (Hash Verified)**

The remainder of this paper is organized as follows: Section II reviews the existing literature and foundational studies. Section III details the system architecture and core components. Section IV presents the implementation and design methodology. Section V evaluates the system's performance and security guarantees. Section VI concludes the paper with future directions.

II. RELATED WORK

Several blockchain-based frameworks have been proposed to mitigate ransomware threats, particularly in high-risk domains like healthcare. Wazid et al. [1], [6] introduced BSFR-SH, a Blockchain-Enabled Security Framework for Smart Healthcare, which integrates distributed ledger mechanisms with role-based access control and anomaly detection to defend against ransomware. The framework ensures secure log preservation and auditability of data access. However, its architecture is tightly coupled to healthcare infrastructure and lacks a decentralized restore mechanism, making it less suitable for general-purpose deployments beyond medical record systems.

Lakhan et al. [3] proposed RBEF, a ransomware-resilient public blockchain architecture that focuses on encrypting and validating digital health data. The system uses smart contracts to control data operations and implements a detection module to flag suspicious activities. While effective in medical use cases, the model does not address decentralized file recovery nor cryptographic validation of restored data. Its dependence on blockchain transaction overhead also poses challenges for latency-sensitive backups.

Delgado-Mohatar et al. [2] examined the concept of semi-autonomous ransomware that operates within blockchain ecosystems, raising questions about how attackers could exploit smart contract logic. Their work is among the first to consider adversarial behavior in decentralized networks. Although insightful, the paper is analytical rather than architectural, and it does not provide a practical solution to ensure the trustworthiness of restored files in hostile environments.

Tahir and Indocyanine [4] proposed a high-level architecture for mitigating ransomware in financial systems using a combination of blockchain, cloud storage, and advanced malware detection techniques. Their model emphasizes fault tolerance and distributed redundancy. However, it lacks implementation specifics, particularly concerning verifiable backup mechanisms and tamper-evident restoration workflows. The absence of technical evaluation limits its application to academic settings.

Vivekrabinson et al. [5] addressed blockchain-backed auditing for electronic health record (EHR) storage by proposing cross-patient block-level deduplication. Their system integrates blockchain to track data changes and detect malicious activity. While useful for maintaining audit trails, it assumes a trusted storage medium and does not protect against ransomware that encrypts or replaces encrypted records at the file level. It also does not support hash-based integrity checking during data recovery.

Ahn et al. [9] developed KEY-SSD, a hardware-level access control system that protects files from unauthorized encryption via a whitelist-based drive controller. The system effectively prevents unknown ransomware from encrypting files by denying I/O access. However, this solution is infrastructure-bound, requiring custom hardware and firmware integration. It is unsuitable for decentralized storage and cannot validate file integrity post-compromise.

Sokolov [7] studied the systemic effects of ransomware on blockchain networks, particularly congestion and fee inflation. His findings provide valuable insight into how ransomware can indirectly degrade blockchain services. While not proposing a defensive architecture, the paper informs the need for lightweight blockchain interactions — an important consideration for backup systems that aim to log or verify frequent file operations on-chain.

Higuchi and Kobayashi [10] presented a real-time backup system using machine learning to detect and counter ransomware. Their model triggers immediate file copying upon detecting encryption-like behavior, minimizing data loss. However, the model lacks file-level verification and cannot prevent the backup of already corrupted or tampered files. Its reactive design contrasts with preventive models that ensure cryptographic integrity at restore time.

Rani et al. [8] conducted a broad survey on ML-based ransomware detection. The paper outlines anomaly detection techniques, classification models, and behavior-based heuristics. While the survey offers a useful taxonomy, it does not propose a complete backup or restore framework and does not address data integrity verification using decentralized mechanisms.

Chandran et al. [11] explored obfuscated malware detection using AI-driven behavioral analysis and digital twin simulation. Their method enhances detection of ransomware variants that evade traditional scanning. However, the solution is purely analytical and lacks a data recovery or validation model.

Achuthan et al. [12] presented a wide-ranging review of AI and privacy-preserving technologies in cybersecurity. They discussed blockchain integration, adversarial machine learning, and federated learning but did not propose an implementation for file-level backup or restore assurance.

Gurram et al. [13] investigated the embedding of malware into deep neural network models, revealing a new frontier of ransomware delivery vectors. While relevant from a threat modeling perspective, their work lacks defense strategies or mitigation through backup verification.

Vinayakumar et al. [14] demonstrated the feasibility of ransomware detection through social media mining using deep learning. Although innovative in triage and alerting, the work does not address data resilience or verifiable recovery post-attack.

Overall, while the reviewed literature provides valuable insights across various aspects of ransomware mitigation — including detection, access control, obfuscation, and blockchain logging — most existing approaches suffer from narrow domain applicability, hardware constraints, or lack verifiable file recovery mechanisms. This motivates our proposed solution, which combines AES-256 encryption, IPFS for decentralized storage, and Ethereum smart contracts to provide secure, auditable, and trustless file backups with end-to-end restore integrity.

III. System Architecture

This section outlines the design of the proposed ransomware-proof backup system, which integrates local encryption, decentralized storage, and smart contract-based verification to enable tamper-evident file restoration. The system is divided into four main modules: the AES Encryption Module, IPFS Storage Module, Smart Contract Module, and Restore Verification Module. Together, these components ensure that files are securely backed up, stored in a decentralized manner, and cryptographically validated before being restored, as illustrated in Fig. 1.

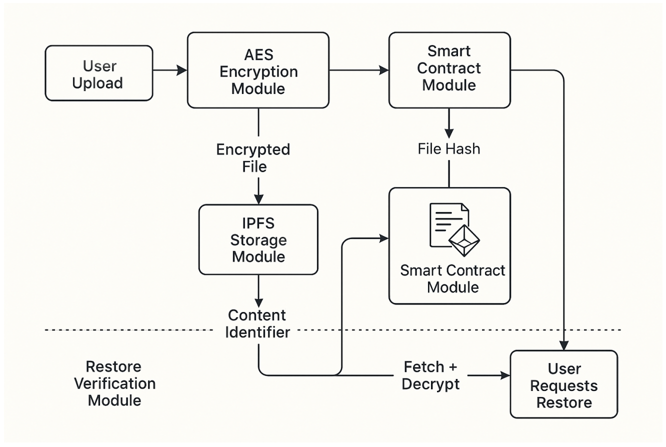


Fig. 1. System architecture of the ransomware-proof backup and restore system.

A. AES Encryption Module

The encryption module is responsible for securing files on the client side before storage. When a user uploads a file, it is encrypted using the AES-256 algorithm in CBC mode with a randomly generated salt and IV. A password-based key derivation function (PBKDF2) is used to generate the encryption key securely. This process ensures data confidentiality and prevents attackers from directly interpreting the contents even if they gain access to the storage layer.

B. IPFS Storage Module

After encryption, the file is uploaded to the InterPlanetary File System (IPFS), a decentralized peer-to-peer storage protocol. IPFS returns a unique Content Identifier (CID) for the uploaded encrypted file. This CID is immutable and acts as a pointer to the exact version of the file stored across the IPFS network. Since IPFS does not rely on a centralized server, it significantly enhances storage availability and resilience against targeted attacks.

C. Smart Contract Module

The CID of the encrypted file is hashed using the SHA-256 algorithm, and the resulting digest is stored immutably on the Ethereum blockchain via a deployed smart contract. This smart contract maintains a mapping of file identifiers (based on filename and timestamp) to their corresponding hashes. Because the blockchain is tamper-proof, this on-chain hash acts as a permanent reference for later file verification during the restoration process.

D. Restore Verification Module

To restore a file, the user requests the system to fetch the encrypted file from IPFS. After decryption using the original AES key, a new hash is computed and compared to the corresponding hash stored on the blockchain. If the hashes match, the file is verified as untampered and is delivered to the user. If there is a mismatch, the restoration process is aborted, preventing corrupted or malicious files from being accepted.

E. Workflow Summary

During backup, each module interacts in sequence: the encryption module secures the file, the storage module handles decentralized distribution, and the blockchain module registers the hash for future verification. Restoration reverses this flow, ending with an integrity check to ensure authenticity.

IV. Implementation and Methodology

This section describes the practical implementation of the ransomware-proof backup system, elaborating on each module introduced in the architecture. The system was developed using Python for the backend logic, IPFS for decentralized storage, Ethereum smart contracts for on-chain hash recording, and Flask for exposing REST APIs. Cryptographic operations are performed using the PyCryptodome library, and blockchain interactions are facilitated through the Web3.py framework. All components are modularly designed to ensure extensibility, security, and real-time file verification capabilities.

A. System Overview

The complete system is implemented as a web-based backup solution with both storage and recovery features. The backend is built using the Flask micro-framework, offering endpoints for file upload and restore operations. The AES encryption logic ensures that files are securely processed on the client side before being exposed to external storage layers. Encrypted files are processed through a coordinated pipeline, wherein IPFS ensures decentralized storage, while Ethereum smart contracts register cryptographic hashes for future verification. This integration enables secure storage, reliable access, and tamper-evident recovery from ransomware events. File restoration follows a reverse path, where the file is downloaded from IPFS, decrypted, and its integrity is verified by recomputing the hash and comparing it with the blockchain-stored record. All user actions are facilitated through a simple web interface or direct API calls using tools like Postman or frontend scripts.

B. AES Encryption Module

To ensure confidentiality prior to file storage, the system employs Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode. Each uploaded file is encrypted locally using a 256-bit AES key derived from a user-supplied password. The key derivation is handled using the Password-Based Key Derivation Function 2 (PBKDF2), which incorporates a randomly generated salt to defend against dictionary and rainbow table attacks.

During encryption, a unique 16-byte Initialization Vector (IV) is also generated for each file session to ensure that identical plaintexts yield different ciphertexts. The salt and IV are prepended to the encrypted file as metadata, enabling accurate decryption later. This design ensures that even if the same file is uploaded twice, the resulting ciphertext and corresponding Content Identifier (CID) from IPFS will be different due to cryptographic entropy.

The PyCryptodome library is used to implement all cryptographic operations, including key generation, padding, encryption, and decryption. File encryption is performed in-memory and written to disk only after cryptographic processing, reducing the attack surface for memory-resident exploits. The resulting '.enc' file is passed to the IPFS module for decentralized storage.

C. IPFS Storage Integration

The encrypted file is uploaded to the InterPlanetary File System (IPFS), a decentralized, content-addressable storage protocol. IPFS ensures file availability by distributing content across a peer-to-peer network, identified by a unique Content Identifier (CID) derived from the file's hash. This CID guarantees immutability, meaning that any alteration in the encrypted content results in a completely different identifier.

File uploads are handled via the IPFS Command Line Interface (CLI), invoked from the backend using secure subprocess calls. Upon successful upload, IPFS returns the CID, which is logged and passed to the blockchain interaction module for hash registration. The system does not rely on any centralized gateway for storage, and the user may choose to pin files persistently using third-party services such as Pinata or Web3.Storage if long-term availability is required.

To preserve privacy, only encrypted files are uploaded, ensuring that no plaintext content is exposed to the network. As a result, the CID corresponds to the encrypted version of the file. The encrypted file, CID, and related metadata are temporarily stored in the backend until the hash is pushed to the blockchain, after which only essential references are retained to reduce attack surface and comply with privacy requirements.

D. Smart Contract and Blockchain Interaction

To ensure the immutability and verifiability of backup records, the system employs an Ethereum smart contract to store the cryptographic hash of the encrypted file. The smart contract is developed in Solidity and deployed on a local Ethereum blockchain using the Hardhat development environment. This approach provides a secure, auditable ledger for storing file integrity references without requiring a public network or incurring real transaction fees.

The smart contract maintains a mapping between a unique file identifier (derived from the filename and timestamp), the raw IPFS CID (for retrieval), and the SHA-3 (Keccak256) hash of the encrypted file's content (for integrity verification). The backend computes this file hash and calls storeBackup() via Web3.py, recording all metadata immutably on-chain.

For retrieval and verification, the 'getBackup()' function is called using the file identifier, allowing the system to fetch the stored hash for comparison during restoration. All blockchain interactions are handled programmatically via Web3.py, including transaction signing, contract deployment, and function execution. To simulate real-world usage and performance, the system will be connected to Ethereum testnets such as Sepolia network.

By leveraging smart contracts, the system guarantees that once a file hash is recorded, it cannot be modified or removed, thereby providing a tamper-proof point of reference for integrity validation during future file restoration.

E. Flask-Based REST API

The application exposes its core functionality through a Flask-based REST API that provides endpoints for secure file upload and restoration. Flask was chosen for its lightweight, modular design and seamless integration with Python-based cryptographic and blockchain libraries.

The /upload endpoint accepts a file and password as input parameters. Upon receiving a request, the backend encrypts the file using AES-256, uploads the encrypted file to IPFS, and computes a Keccak-256 (SHA-3) hash of the encrypted file's contents. This hash, along with the file's CID and a generated file identifier, is recorded on the Ethereum blockchain using a smart contract. The API then returns the CID, transaction hash, and file identifier.

The '/restore' endpoint accepts a file identifier and password. It retrieves the corresponding CID from the blockchain, fetches the encrypted file from IPFS, and decrypts it using the password-derived key. A new hash is computed and compared with the on-chain hash for verification. If the hashes match, the file is returned to the user; otherwise, an integrity warning is issued and restoration is aborted.

Error handling is implemented at every step to manage exceptions such as incorrect passwords, invalid file identifiers, IPFS failures, and blockchain transaction issues. Response codes are standardized (e.g., 200, 400, 403, 500) to support integration with web clients or automation tools such as Postman. Flask-CORS is used to enable frontend-backend interaction if needed in future extensions.

F. Hashing and Verification Process

A central feature of the proposed system is its ability to detect tampering during the restoration process through cryptographic hash verification. After a file is encrypted and uploaded to IPFS, the system generates a Keccak-256 (SHA-3) hash of the encrypted file's contents. This hash serves as a fingerprint and is immutably recorded on the blockchain via the smart contract. During restoration, the file is re-downloaded and hashed again to verify its integrity against the on-chain value.

During the restoration process, the system retrieves the CID corresponding to the file identifier from the smart contract using the 'getBackup()' function. The encrypted file is fetched from IPFS and decrypted using the original AES key derived from the user-provided password. Once decrypted, the system recomputes the SHA-256 hash of the retrieved encrypted CID and compares it with the CID stored on the blockchain.

If the recomputed hash matches the on-chain hash, the file is considered authentic and is returned to the user. Otherwise, the process is halted, and the user is notified of potential tampering or corruption. This mechanism ensures that even if the IPFS-stored file is altered, the system will detect the inconsistency before completing the restore, thereby offering a tamper-evident restoration pipeline.

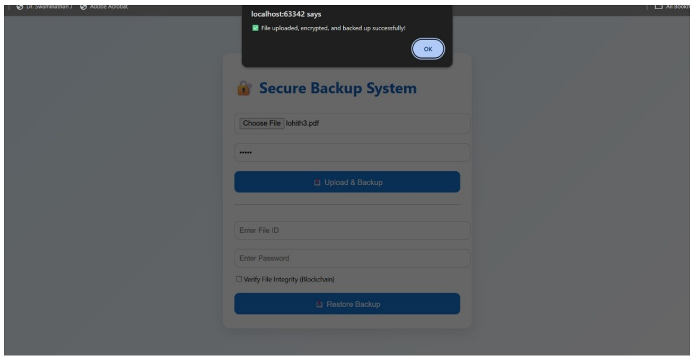
The use of SHA-256, a collision-resistant cryptographic hash function, guarantees that even a single-byte change in the file or CID will result in a completely different hash, making this method highly reliable for data integrity verification.

V. Results and Evaluation

A. Functional Testing

To validate the system's end-to-end functionality, a series of test cases were executed covering the core operations: file encryption, IPFS upload, blockchain hash storage, and integrity-verified restoration. The tests confirm that the system performs as expected under normal, untampered conditions.

During the upload phase, a sample file is encrypted using AES-256-CBC and successfully uploaded to IPFS via the backend interface. Upon successful upload, the backend returns a Content Identifier (CID) along with a blockchain transaction hash indicating that the file's SHA-256 hash has been recorded immutably using the deployed smart contract.


localhost:63342 says

✅ File uploaded, encrypted, and backed up successfully!

Fig. 2. Sample API response showing CID and transaction hash after secure upload.

The corresponding file identifier and password are then used to trigger the restoration process. The system retrieves the CID from the smart contract using the identifier, downloads the file from IPFS, and decrypts it locally. A new hash is computed and compared with the on-chain record.

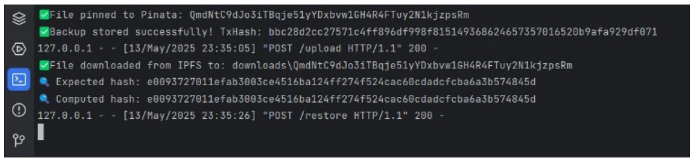

```
✅File pinned to Pinata: QmdNtC9dJo3iTBqje51yYDxbvw1GH4R4FTuy2N1kjzpsRm
✅Backup stored successfully! TxHash: bbc28d2cc27571c4ff896df998f8151493686246573570165 20b9afa929df071
127.0.0.1 - - [13/May/2025 23:35:05] "POST /upload HTTP/1.1" 200 -
✅File downloaded from IPFS to: downloads\QmdNtC9dJo3iTBqje51yYDxbvw1GH4R4FTuy2N1kjzpsRm
🔍 Expected hash: e0093727011efab3003ce4516ba124ff274f524cac60cdadcfcba6a3b574845d
🔍 Computed hash: e0093727011efab3003ce4516ba124ff274f524cac60cdadcfcba6a3b574845d
127.0.0.1 - - [13/May/2025 23:35:26] "POST /restore HTTP/1.1" 200 -
```

Fig. 3. Integrity check passed: file restored after hash match with blockchain record.

The hash comparison succeeded in all valid test cases, confirming that the restoration pipeline is functioning correctly. Files were returned only when their decrypted hash matched the immutably stored blockchain hash, thus validating the system's core objective of trustless, verifiable restoration.

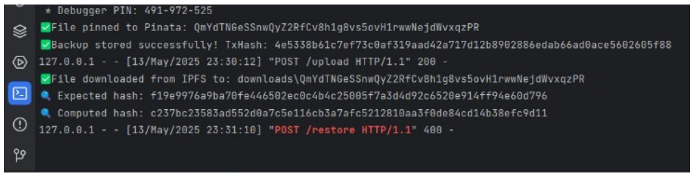
B. Tamper Detection and Verification

A critical feature of the proposed system is its ability to detect file tampering during the restoration process. To evaluate this, test cases were designed where either the encrypted file stored on IPFS or the associated blockchain hash was intentionally modified.

In the first scenario, the original encrypted file was replaced with a corrupted version at the same CID location. When the '/restore' endpoint was triggered using the correct file identifier and password, the system retrieved the corrupted file from IPFS and decrypted it. Upon recomputing the hash and comparing it with the blockchain-stored hash, a mismatch was detected. The system immediately aborted the restoration and returned an integrity error response.

In the second scenario, the file hash stored on the blockchain was altered (via manual contract redeployment for test purposes). Even though the file on IPFS remained unchanged, the recomputed hash at restore time did not match the tampered on-chain hash, and the restoration was rejected.

Both test cases demonstrate that the system successfully detects unauthorized modifications at both the storage layer



```
 * Debugger PIN: 491-972-525
✅File pinned to Pinata: QmYdTNGeSSnwQyZ2RfCv8h1g8vs5ovH1rwwNejdWvxqzPR
✅Backup stored successfully! TxHash: 4e5338b61c7ef73c0af319aad42a717d12b8902886edab66ad0ace5602605f88
127.0.0.1 - - [13/May/2025 23:30:12] "POST /upload HTTP/1.1" 200 -
✅File downloaded from IPFS to: downloads\QmYdTNGeSSnwQyZ2RfCv8h1g8vs5ovH1rwwNejdWvxqzPR
🔍 Expected hash: f19e9976a9ba70fe446502ec0c4b4c25005f7a3d4d92c6520e914ff94e60d796
🔍 Computed hash: c237bc23583ad552d0a7c5e116cb3a7afc5212810aa3f0de84cd14b38efc9d11
127.0.0.1 - - [13/May/2025 23:31:10] "POST /restore HTTP/1.1" 400 -
```

Fig. 4. Integrity check failed: hash mismatch detected between IPFS file and on-chain record.

and verification layer. This confirms the integrity validation mechanism is functioning as intended and prevents the restoration of potentially malicious or corrupted files.

C. Security Evaluation

The proposed system was analyzed against common ransomware attack vectors to assess its robustness from a security standpoint. Traditional ransomware exploits include encrypting or deleting local files, targeting backup folders, and modifying logs or metadata to obscure attack traces. Centralized backup systems are especially vulnerable due to their reliance on trusted servers and lack of tamper-evident mechanisms.

In contrast, the system presented in this work leverages a decentralized storage and validation model that removes these single points of failure. By encrypting files locally using AES-256 before uploading them to IPFS, the system ensures that no readable data is exposed to the storage network, effectively neutralizing ransomware attempts to exfiltrate or analyze data contents. The use of PBKDF2 and unique IVs per session further mitigates brute-force and replay attacks.

Integrity protection is achieved through SHA-256 hashing and blockchain anchoring. Once the hash of the encrypted file is stored on the Ethereum smart contract, it becomes immutable and verifiable. Ransomware attempting to replace or overwrite backup data on IPFS would cause the hash comparison to fail at restore time, thus blocking the attack's success.

Furthermore, the restore process is conditioned on hash verification. This ensures that even if a malicious actor gains control over the IPFS node or backend server, any unauthorized changes to the backup files will be detected. The system thereby resists ransomware behaviors such as stealthy encryption of backups or substitution of poisoned data, making it significantly more resilient than conventional backup architectures.

D. Summary of Observations

The results of functional and adversarial testing confirm the effectiveness and reliability of the proposed ransomware-proof backup system. Under normal operating conditions, the system successfully encrypted files, uploaded them to IPFS, and recorded immutable hash references on the blockchain. Restoration was consistently successful when file integrity was preserved.

In contrast, when files were intentionally tampered with—either by altering the encrypted file on IPFS or modifying the on-chain hash—the system detected mismatches and prevented restoration. This demonstrates the reliability of the hash verification process and validates the integrity-check mechanism as a critical safeguard against unauthorized modifications.

Security evaluation further reinforces that the system mitigates key ransomware tactics, including backup compromise, stealth encryption, and data substitution. The combination of local encryption, decentralized storage, and immutable on-chain validation offers a multi-layered defense model not present in traditional backup solutions.

On average, the end-to-end upload process — including encryption, IPFS storage, and blockchain hash registration — was completed in under 5 seconds for files up to 1MB in size, validating the system's practical feasibility for real-time usage.

Overall, the system meets its core design goals: ensuring secure, decentralized storage of encrypted backups; enabling tamper-evident restoration; and providing robust protection against ransomware-induced data loss. The implementation exhibits practical feasibility, strong security guarantees, and extensibility for future enhancements.

VI. Conclusion and Future Work

This paper presented a ransomware-proof backup system that integrates AES-256 encryption, decentralized IPFS storage, and blockchain-based hash validation. The system ensures tamper-evident backups and trustless file restoration through cryptographic verification.

Experimental testing confirmed correct functionality under normal and adversarial conditions, including successful detection of tampered files. By eliminating centralized trust and enforcing on-chain integrity checks, the system offers enhanced resilience against ransomware threats.

Future work may explore versioned backups, UI integration, and support for permissioned blockchains to improve scalability and usability.

References

[1] Wazid, M., Das, A., & Shetty, S., 2023. BSFR-SH: Blockchain-Enabled Security Framework Against Ransomware Attacks for Smart Healthcare. IEEE Transactions on Consumer Electronics, 69, pp. 18-28.

[2] Delgado-Mohatar, O., Camara, J., & Anguiano, E., 2020. Blockchain-based semi-autonomous ransomware. Future Gener. Comput. Syst., 112, pp. 589-603.

[3] Lakhan, A., Thinnukool, O., Grønli, T., & Khuwuthyakorn, P., 2023. RBEF: Ransomware Efficient Public Blockchain Framework for Digital Healthcare Application. Sensors (Basel, Switzerland), 23.

[4] Tahir, F. and Indocyanine, W., 2024. Future-Proofing Financial Institutions: Mitigating Ransomware with Blockchain, Cloud Computing, and Advanced Malware Prevention.

[5] Vivekrabinson, K., Ragavan, K., Jothi Thilaga, P. and Bharath Singh, J., 2024. Secure cloud-based electronic health records: Cross-patient block-level deduplication with blockchain auditing. Journal of Medical Systems, 48(1), p.33.

[6] Wazid, M., Das, A.K. and Shetty, S., 2022. BSFR-SH: Blockchain-enabled security framework against ransomware attacks for smart healthcare. IEEE Transactions on Consumer Electronics, 69(1), pp.18-28.

[7] Sokolov, K., 2021. Ransomware activity and blockchain congestion. Journal of Financial Economics, 141(2), pp.771-782.

[8] Kirubavathi, G., Regis Anne, W. and Sridevi, U.K., 2024. A recent review of ransomware attacks on healthcare industries. International Journal of System Assurance Engineering and Management, 15(11), pp.5078-5096.

[9] J. Ahn, D. Park, C.-G. Lee, D. Min, J. Lee, S. Park, Q. Chen, and Y. Kim, "KEY-SSD: Access-control drive to protect files from ransomware attacks" arXiv preprint arXiv:2206.11500, 2022.

[10] Higuchi, K. and Kobayashi, R., 2025. Real-time open-file backup system with machine-learning detection model for ransomware. International Journal of Information Security, 24(1), p.54.

[11] Chandran, S., Syam, S.R., Sankaran, S., Pandey, T. and Achuthan, K., 2025. From Static to AI-Driven Detection: A Comprehensive Review of Obfuscated Malware Techniques. IEEE Access.

[12] Achuthan, K., Ramanathan, S., Srinivas, S. and Raman, R., 2024. Advancing cybersecurity and privacy with artificial intelligence: current trends and future research directions [Review of Advancing cybersecurity and privacy with artificial intelligence: current trends and future research directions]. Frontiers in Big Data, 7. Frontiers Media. https://doi. org/10.3389/fdata.

[13] Gurram, V.R., Amritha, P.P. and Sethumadhavan, M., 2023, September. Exploiting Neural Network Model for Hiding and Triggering Malware. In World Conference on Information Systems for Business Management (pp. 209-220). Singapore: Springer Nature Singapore.

[14] Vinayakumar, R., Alazab, M., Jolfaei, A., Soman, K.P. and Poornachandran, P., 2019, May. Ransomware triage using deep learning: twitter as a case study. In 2019 Cybersecurity and Cyberforensics Conference (CCC) (pp. 67-73). IEEE.