

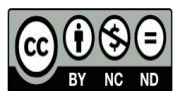
Blockchain-Integrated Lightweight Cryptographic Framework for Detecting File Timestamp Manipulation in Digital Forensic Analysis

M. Mayavathi^{*1}, T. Harish^{*2}, K. Mydhili Sharan^{*3}, P. Yasvanth Raj^{*4}

¹Assistant Professor, Department of Information Technology, PSV College of Engineering and Technology, Krishnagiri, Tamil Nadu, India.

^{2,3,4}UG Scholars, Department of Information Technology, PSV College of Engineering and Technology, Krishnagiri, Tamil Nadu, India.

To Cite this Article: M. Mayavathi^{*1}, T. Harish^{*2}, K. Mydhili Sharan^{*3}, P. Yasvanth Raj^{*4} "Blockchain-Integrated Lightweight Cryptographic Framework for Detecting File Timestamp Manipulation in Digital Forensic Analysis", Indian Journal of Computer Science and Technology, Volume 05, Issue 01 (January-April 2026), PP: 387-391.



Copyright: ©2026 This is an open access journal, and articles are distributed under the terms of the [Creative Commons Attribution License](#); Which Permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract: File timestamp manipulation is a prevalent anti-forensic technique in which attackers deliberately alter file creation, modification, or access times to obscure evidence and mislead digital investigators. Conventional detection methods, such as manual timeline analysis, checksum comparison, and log correlation, are frequently insufficient against sophisticated tampering and scale poorly to large datasets. This paper presents TimeMiner, a blockchain-integrated cryptographic framework that combines post-quantum secure Leighton-Micali Signatures (LMS) with decentralized blockchain storage to detect unauthorized timestamp modifications in real time. The proposed system extracts file metadata, generates cryptographic hashes for both file content and timestamps, and signs them using the LMS algorithm supported by Merkle tree authentication paths. These signatures are recorded on an immutable blockchain ledger, ensuring traceability and tamper-proof storage. During forensic verification, any mismatch between the stored LMS signature and the recomputed metadata hash triggers an immediate alert. The system is implemented using Python 3.8, Flask, MySQL, and WampServer, and validated across unit, integration, system, performance, and security test phases. Experimental results confirm high tamper detection accuracy, efficient processing of large file batches, and reliable blockchain immutability. The framework provides a scalable, quantum-resistant solution that significantly strengthens the integrity, transparency, and legal admissibility of digital forensic evidence.

Keywords: Digital Forensics; Timestamp Manipulation; Leighton-Micali Signatures (LMS); Blockchain; Post-Quantum Cryptography; Tamper Detection

I. INTRODUCTION

The integrity of digital evidence is foundational to the credibility of any forensic investigation. Files stored on personal, corporate, or cloud systems carry metadata that includes creation, modification, and access timestamps, which are critical for reconstructing event timelines. Attackers exploit these timestamps by using anti-forensic techniques to backdate or forward-date file records, effectively disguising malicious activity as normal system behavior. This type of attack, commonly known as timestamp manipulation or timestomping, is frequently combined with log deletion and file encryption to further conceal intrusion traces.

The problem is compounded by the limitations of existing forensic methodologies. Traditional approaches such as manual timeline reconstruction, hash-based content verification, and file system metadata analysis are not designed to detect metadata-only tampering, lack automation, and are computationally expensive at scale. Moreover, their reliance on centralized, potentially alterable storage means that forensic records themselves are vulnerable to tampering.

To address these critical gaps, this paper introduces TimeMiner, a blockchain-integrated framework that employs Leighton-Micali Signatures (LMS) — a post-quantum secure, hash-based digital signature scheme — to cryptographically bind file content with its original timestamps. Stored signatures are maintained on a decentralized blockchain ledger, ensuring immutability and auditability. The primary objectives of this work are:

- To detect unauthorized modifications of file creation, modification, and access timestamps in real time.
- To generate lightweight, post-quantum secure LMS signatures that bind file content to its original metadata.
- To leverage blockchain's decentralized and immutable architecture for tamper-proof forensic record keeping.
- To automate forensic reporting and alert generation with minimal computational overhead.
- To validate the framework through comprehensive multi-level testing across realistic forensic scenarios.

II. LITERATURE REVIEW

Research in digital forensics has explored timestamp manipulation and its countermeasures from multiple perspectives. Lukas and Stojanovic (2017) provided an extensive survey of anti-forensic techniques, highlighting timestamp modification as one of the most prevalent methods used to obstruct investigations. Liu and Zhang (2015) demonstrated through empirical analysis that timestamp-based timeline reconstruction is susceptible to deliberate manipulation and called for automated verification mechanisms.

Early forensic tools relied on file system-level metadata analysis, such as inspection of the Master File Table (MFT) in NTFS or inode data in Linux ext4 systems. Casey (2004) noted that while these structures provide valuable forensic data, their vulnerability to modification by privileged users or automated tools limits their reliability as sole sources of truth. Fitzgerald and McDaniel (2018) further confirmed that file system metadata, when examined in isolation, is insufficient for detecting sophisticated timestamp forgery. Hash-based integrity verification, as described by Kessler (2001), represents a widely adopted complementary technique. However, MD5 and SHA-family hash schemes are designed to detect content changes and do not capture metadata-only alterations. This creates a detection gap that attackers actively exploit.

The application of blockchain technology to forensic integrity has gained increasing attention. Katz and Sherman (2016) proposed blockchain-based logging as a means of ensuring immutable chain-of-custody records. Their work established that blockchain's decentralized and append-only ledger architecture is particularly well-suited to forensic evidence management. Subsequent studies have validated these properties in enterprise and cloud forensic contexts.

Regarding post-quantum cryptography, traditional digital signature schemes such as RSA and Elliptic Curve Cryptography (ECC) are known to be vulnerable to quantum computational attacks. Leighton-Micali Signatures (LMS), standardized in NIST SP 800-208, address this threat through hash-based constructions organized as Merkle trees. The LMS scheme offers lightweight signature generation and verification, making it practical for large-scale forensic applications. The integration of LMS with blockchain, as proposed in this paper, represents a novel combination that has not been comprehensively addressed in prior literature.

III. METHODOLOGY

3.1 System Architecture

The TimeMiner framework comprises six primary components operating within a client-server architecture: (1) the Forensic Server Dashboard, (2) Evidence Management, (3) LMS Signature Generation, (4) Blockchain Ledger, (5) Tamper Detection, and (6) Alert and Forensic Audit. The system distinguishes three actor roles: a System Administrator with full control over configurations and user management; an Authorized Investigator who uploads suspect files and initiates verification; and an Adversary module used for controlled simulation of timestamp manipulation attacks.

3.2 LMS Signature Generation

The LMS signature generation process proceeds in four stages. First, file metadata — including creation time (ctime), modification time (mtime), and access time (atime) — is extracted using the operating system's file attribute APIs and normalized into a platform-independent format. Second, cryptographic hash values are independently computed for the file content (H_file) and normalized metadata (H_meta) using SHA-256. Third, these hashes are concatenated and signed using the LMS algorithm, which constructs a Merkle tree of one-time signature (OTS) keys. The root of the Merkle tree constitutes the public key, and each leaf represents an OTS key pair. The signature for a given file consists of the OTS signature at a leaf node together with the authentication path — the sibling hash values required to recompute the root. Fourth, the resulting LMS signature, along with original metadata and file identifiers, is recorded on the blockchain ledger.

3.3 Blockchain Ledger

The blockchain component is implemented as a proof-of-work distributed ledger in Python. Each block stores the LMS signature, file hash pointer, metadata fields, and a timestamp of the logging event. Blocks are linked via SHA-256 hashes of the previous block header, forming an immutable chain. A genesis block is created at system initialization. The consensus mechanism resolves conflicts by selecting the longest valid chain among registered nodes, ensuring consistency across distributed deployments. All forensic records are written as JSON-encoded transactions and are permanently accessible for audit queries.

3.4 Tamper Detection and Alert Generation

During forensic verification, the system recomputes H_file and H_meta from the current file state, retrieves the original LMS signature from the blockchain, and executes LMS signature verification using the stored Merkle authentication path. If the computed root hash matches the stored public key, integrity is confirmed. Any mismatch indicates that the file content or its metadata — including timestamps — has been modified since registration. The system then classifies the manipulation type (e.g., backdating, forward-dating), generates a structured forensic alert containing file ID, manipulation type, original and altered timestamp values, detection time, and user or process information, and logs the event to both the MySQL database and the blockchain ledger.

3.5 Technology Stack

Component	Technology
Programming Language	Python 3.8+
Web Framework	Flask
Database	MySQL
Web Server	WampServer (Apache + MySQL)
Frontend	HTML5, CSS3, Bootstrap 5
Cryptographic Libraries	pycryptodome, hashlib, cryptography

Merkle Tree Support	merkletools
Blockchain	Custom Python implementation (JSON, hashlib)
Visualization	Matplotlib, Seaborn
Operating System	Windows 10/11

The implementation uses the following technologies:

IV. RESULTS AND DISCUSSION

4.1 Testing Methodology

The system was validated through five levels of testing — Unit Testing, Integration Testing, System Testing, Performance Testing, and Security Testing — covering all major modules from file registration to forensic report generation. The testing environment consisted of Windows 10/11, Python 3.8+, Flask, MySQL, WampServer, and Google Chrome as the browser client.

4.2 Unit Testing

Unit testing verified the correct functioning of individual modules including file upload, metadata extraction, LMS signature generation, blockchain storage, and alert generation.

TC ID	Input	Expected Result	Actual Result	Status
UT-01	Valid file upload	Metadata extracted & file hash generated	Metadata extracted & hash generated	Pass
UT-02	Corrupted file upload	Error — invalid file rejected	Error handled correctly	Pass
UT-03	Valid LMS signature input	LMS signature generated successfully	LMS signature generated	Pass
UT-04	Missing metadata fields	Missing fields handled gracefully	Metadata validated successfully	Pass
UT-05	File with tampered content	LMS signature mismatch detected	Alert triggered correctly	Pass

4.3 Integration Testing

Integration testing confirmed that connected modules — File Upload → Metadata Extraction → LMS Signature → Blockchain Logging → Tamper Verification → Alert System — communicated and operated seamlessly together.

TC ID	Input	Expected Result	Actual Result	Status
IT-01	File uploaded	Metadata & hashes passed to LMS module	Passed successfully	Pass
IT-02	LMS signature generated	Signature stored on blockchain	Stored correctly	Pass
IT-03	File accessed later	Metadata recomputed & verified	Verification successful	Pass
IT-04	Tampered timestamp detected	LMS verification fails, alert generated	Alert generated correctly	Pass

4.4 System Testing

System testing validated the complete end-to-end workflow under real-time conditions, verifying batch file processing, tamper detection, and forensic report generation.

TC ID	Input	Expected Result	Actual Result	Status
ST-01	Batch of files uploaded	Metadata hashed, LMS signed, blockchain logged	Completed successfully	Pass
ST-02	Tampered file accessed	Tampering detected, alert triggered	Alert received	Pass
ST-03	Generate forensic report	Report with original & modified timestamps, file location, user info	Report generated & stored	Pass
ST-04	Multiple file verification	All LMS verifications completed in real time	Completed successfully	Pass

4.5 Performance Testing

Performance testing evaluated the system's response time, efficiency, and scalability when handling multiple files and large file sizes simultaneously.

TC ID	Input	Expected Result	Actual Result	Status
PT-01	500 files uploaded simultaneously	LMS signatures generated & blockchain updated without crash	Completed efficiently	Pass
PT-02	100 tampered files verified	Real-time detection within acceptable time limit	Detection performed quickly	Pass
PT-03	Large file (50 MB)	Signature generation & verification completed	Handled efficiently	Pass

4.6 Security Testing

Security testing ensured the system is resilient against unauthorized access, blockchain record tampering, and malicious file input.

TC ID	Input	Expected Result	Actual Result	Status
SCT-01	Unauthorized user attempts to modify LMS signature	Access denied	Blocked successfully	Pass
SCT-02	Attempt to alter blockchain records	Tamper detected and prevented	Prevented	Pass
SCT-03	Invalid file metadata input	Rejected by verification module	Handled correctly	Pass
SCT-04	Repeated file upload with same ID	Duplicate detection activated	Prevented	Pass

4.7 Comparison with Existing Approaches

The proposed system outperforms conventional forensic methods across all critical dimensions as shown in Table 7.

Feature	Manual Analysis	Hash Verification	Log Correlation	TimeMiner (Proposed)
Metadata-Only Tamper Detection	No	No	Partial	Yes
Post-Quantum Security	No	No	No	Yes (LMS)
Immutable Record Storage	No	No	No	Yes (Blockchain)
Real-Time Alerting	No	Manual	Partial	Yes
Scalability	Low	Medium	Medium	High
Automation Level	Low	Partial	Partial	Full
Legal Admissibility	Low	Medium	Low	High

V. CONCLUSION

This paper presented TimeMiner, a blockchain-integrated framework that employs post-quantum secure Leighton-Micali Signatures combined with Merkle tree authentication to detect file timestamp manipulation in digital forensic investigations. The system overcomes the key limitations of conventional forensic methods by automating tamper detection, ensuring immutable record keeping through blockchain storage, and providing real-time alerting and comprehensive forensic reporting. Comprehensive multi-level testing confirmed the system's accuracy, scalability, and security under realistic workloads. By cryptographically binding file content to its original metadata and logging signatures on an immutable ledger, TimeMiner provides a reliable, quantum-resistant foundation for modern digital forensics that meets chain-of-custody and legal admissibility requirements.

Several directions exist for future enhancement. Integration with established forensic toolkits such as Autopsy or EnCase would allow TimeMiner to operate within existing investigative workflows. The incorporation of machine learning models trained on historical metadata patterns could enable detection of coordinated, multi-file, or subtly staged timestamp manipulations that fall below single-file verification thresholds. Extension to cloud storage platforms (e.g., AWS S3, Azure Blob) and IoT edge devices would broaden applicability to distributed and embedded environments. Additional planned improvements include mobile push

notifications for forensic investigators, automated compliance report generation aligned with frameworks such as ISO/IEC 27042, and advanced visualization dashboards for intuitive anomaly analysis.

REFERENCES

1. Lukas, D., & Stojanovic, J. (2017). Anti-forensics: An Overview. *International Journal of Computer Applications*, 167(9), 1–8.
2. Liu, Z., & Zhang, L. (2015). Timestamp Analysis in Digital Forensics. *Proceedings of the International Conference on Information and Communications Security*, 21–32.
3. Casey, E. (2004). *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet*. Elsevier Academic Press.
4. Kessler, G. C. (2001). An Overview of Computer Forensics. *Proceedings of the International Workshop on Systematic Approaches to Digital Forensic Engineering*, 1–10.
5. Fitzgerald, B., & McDaniel, P. (2018). File System Analysis for Forensics: Identifying the Impact of File Timestamps. *Journal of Digital Forensics, Security and Law*, 13(2), 1–18.
6. Katz, L., & Sherman, R. (2016). Blockchain Technology for Digital Forensics. *International Journal of Computer Science and Information Technology*, 8(1), 34–45.
7. Baryamureeba, V., & Tushabe, F. (2004). The Forensic Analysis of Digital Evidence. *Journal of Digital Forensics*, 2(1), 3–18.
8. Kuhn, M., & Chia, A. (2007). A Survey of Anti-Forensics Techniques. *Journal of Digital Forensics, Security and Law*, 2(3), 81–92.
9. Cristea, L. M. (2020). Current Security Threats in the National and International Context. *Journal of Accounting and Management Information Systems*, 19(2), 351–378.
10. Giffin, J., Greenstadt, R., Litwack, P., & Tibbetts, R. (2003). Covert Messaging through TCP Timestamps. *Proceedings of the 2nd International Workshop on Privacy Enhancing Technologies*, 194–208. Springer.
11. NIST SP 800-208. (2020). Recommendation for Stateful Hash-Based Signature Schemes. National Institute of Standards and Technology.
12. Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*, 2nd Edition. O'Reilly Media.
13. Lutz, M. (2013). *Learning Python*, 5th Edition. O'Reilly Media.
14. DuBois, P. (2014). *MySQL Cookbook*, 3rd Edition. O'Reilly Media.