ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 01, January 2026

SmartConSecurity: An SQLite RDBMS with Blockchain Technology Using SHA-256 Hashing Algorithm with a Salting Process

[1] John Anthony E. Torrejas, [2] Dr. Godwin S. Monserate

[1] [2] Department of Computer, Information Sciences, and Mathematics, University of San Carlos, Philippines
Corresponding Author Email: [1] 18102597@usc.edu.ph, [2] gsmonserate@usc.edu.ph

***Abstract*— Data transparency and integrity remain critical challenges in centralized institutions, where clients often lack visibility into how their data is safeguarded against tampering. To address these challenges, this study presents SmartConSecurity, a framework that combines salted SHA-256 (Secure Hash Algorithm 256-bit) hashing, AES-256-CTR (Advanced Encryption Standard, 256-bit key, in Counter mode) encryption, and blockchain storage to provide verifiable data integrity and transparency. A session-based salting mechanism ensures that even identical files generate unique hashes and ciphertexts, preventing duplication and strengthening tamper resistance. A lightweight web application integrated with MetaMask facilitates BSC (Binance Smart Chain) blockchain interaction, while a watchdog module continuously monitors files in real time to detect unauthorized modifications. To ensure confidentiality, the salted hash also serves as the AES-256- CTR encryption key, with the corresponding nonce stored on-chain for secure verification. Cryptographic validation confirms robustness: salted hashing produced an average 48.2%-bit difference (Hamming distance), randomness tests met NIST (National Institute of Standards and Technology) standards, and collision probability remained negligible ($<10^{-54}$ for 10^{12} records). Performance evaluation showed that SmartConSecurity handles files up to 100 MB, maintaining end-to-end encryption and decryption within practical time bounds. These results demonstrate that SmartConSecurity provides a practical, scalable approach for transforming institutional data management from a trust-based to a verifiable model.**

***Index Terms*— aes-256 encryption (ctr mode), blockchain technology, data integrity, sha-256 hashing algorithm, verifiable database.**

I. INTRODUCTION

Data integrity ensures the validity and reliability of stored information and is critical in domains such as education, banking, and healthcare. As cyberattacks increase in frequency and sophistication, ensuring that digital records remain untampered has become a persistent challenge. Among the most prevalent threats are SQL (Structured Query Language) injection attacks, which can lead to unauthorized and often undetected modifications of sensitive data [1], [2], [3].

Cryptographic hash functions such as SHA-256 are widely used to protect data integrity by generating unique digital fingerprints [4]. However, when both data and its corresponding fingerprint are stored within the same database, integrity guarantees can be undermined if an attacker gains sufficient privileges [3], [5]. Prior studies have shown that SQL injection attacks can simultaneously alter records and their associated hashes, leaving no detectable evidence of tampering [5].

Blockchain technology has been proposed as a solution by providing immutable, distributed storage for integrity proofs [6]. By anchoring cryptographic fingerprints on a blockchain, unauthorized database modifications can be detected through cross-verification. However, the transparency of public blockchain networks introduces new risks, as exposed fingerprints may be subjected to brute-force reconstruction attempts [7]. This limits the effectiveness of blockchain-only approaches for sensitive integrity verification.

Institutions that rely on centralized databases, therefore, face a dual challenge: protecting sensitive data from tampering while also enabling clients to independently verify data integrity. Existing integrity mechanisms often operate as opaque, institution-controlled processes, leaving users unable to verify whether their data remains intact or whether institutional claims can be trusted. While prior studies have explored blockchain-backed audit logs and hybrid encryption-hashing schemes, these approaches frequently treat confidentiality and verifiable transparency as separate objectives.

To address this gap, this study proposes SmartConSecurity, a framework that integrates salted SHA-256 hashing, AES-256-CTR encryption, and blockchain-backed verification to provide confidentiality, tamper resistance, and verifiable data integrity within a unified system. By combining session-based salting, encrypted storage, and real-time monitoring, SmartConSecurity transforms institutional data management from a trust-based model to a verifiable one.

II. MOTIVATION AND OBJECTIVES

Centralized database systems place institutions in full control of data storage, integrity verification, and security reporting. While such systems are widely adopted for their efficiency and scalability, they require clients to trust that institutions correctly safeguard their data and promptly detect

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 10, Issue 2, February 2023

unauthorized modifications. In practice, this trust-based model is vulnerable to insider threats, SQL injection attacks, and silent data manipulation, where records and associated integrity checks may be altered without detection [5], [8].

Existing integrity mechanisms offer only partial solutions. Hash-based verification can detect changes, but fails when hashes and data are stored together. Blockchain-based approaches provide immutability but expose integrity fingerprints to public visibility, introducing confidentiality and brute-force risks [7]. Hybrid cryptographic solutions strengthen confidentiality but often lack independent verification and real-time monitoring. These limitations highlight the absence of a unified framework that enables both secure data storage and transparent, client-controlled integrity verification.

The motivation of this work is to shift institutional data management from a trust-based paradigm to a verify-based paradigm, where clients can independently validate the integrity of their data without relying on institutional claims. Achieving this requires a system that preserves confidentiality, prevents replay or duplication attacks, anchors verification data in an immutable medium, and detects unauthorized modifications as they occur.

Accordingly, the objectives of this study are as follows:

1) ***Enable dynamic and non-repeating integrity verification:*** by ensuring that identical files uploaded across different sessions always produce unique cryptographic fingerprints.
2) ***Preserve data confidentiality:*** by encrypting stored files while binding integrity verification directly to the encryption process.
3) ***Provide immutable and independently verifiable integrity references:*** through blockchain-backed anchoring of verification metadata.
4) ***Detect and respond to unauthorized modifications in real time:*** by alerting data owners immediately upon integrity violations.

These objectives guide the design of SmartConSecurity and inform the architectural and methodological choices presented in the succeeding sections.

III. CONTRIBUTION STATEMENT AND METHODOLOGY

This section presents the core contributions of SmartConSecurity and describes the methodology used to implement the proposed framework. The design integrates cryptographic mechanisms, blockchain-based verification, and real-time monitoring to provide a unified approach to secure and verifiable data management

A. Contribution Statement

SmartConSecurity introduces the following key contributions:

1) ***Session-Based Salting for Dynamic Integrity Verification*:** A session-based salting mechanism is employed to ensure that identical files uploaded across different sessions always produce distinct cryptographic fingerprints. This prevents replay attacks, eliminates hash duplication, and strengthens resistance against precomputation and rainbow-table attacks [9], [10].
2) ***Unified Cryptographic Pipeline for Confidentiality and Integrity*:** The framework integrates salted SHA-256 hashing with AES-256-CTR encryption [9], [11]. Where the salted hash itself serves as the encryption key. This tightly binds integrity verification and confidentiality, ensuring that any tampering invalidates both the decrypted data and its verification reference.
3) ***Blockchain-Backed Immutable Verification Anchoring*:** SmartConSecurity uses a lightweight smart contract deployed on the BNB Smart Chain to immutably store salted hashes and encryption nonces [12]. This provides an independently verifiable integrity reference that remains tamper-resistant, while avoiding the overhead of storing full data on-chain.
4) ***Real-Time Tamper Detection via Watchdog Monitoring*:** An event-driven watchdog mechanism continuously monitors encrypted database files. Upon detecting unauthorized modifications, the system automatically verifies integrity against blockchain records and immediately alerts the data owner, enabling a real-time response to tampering.

B. System Architecture

The SmartConSecurity framework consists of two primary components: a web-based application and a blockchain network. These components interact to separate encrypted data storage from immutable integrity verification.

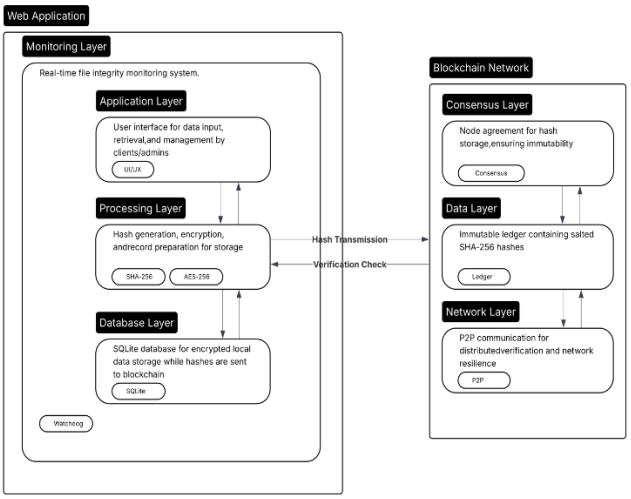

Figure 1. SmartConSecurity Two-Component Architecture

The web application component manages user interaction, encryption, verification, and monitoring. It includes an application layer for file management, a processing layer for cryptographic operations, a monitoring layer for real-time

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 10, Issue 2, February 2023

integrity checks, and a local SQLite database for encrypted storage.

The blockchain network component serves as an immutable verification layer. A smart contract manages per-user hash identifiers and stores salted hashes and encryption nonces. Distributed consensus ensures that verification metadata cannot be altered without detection.

C. Methodology and Workflow Logic

To operationalize the proposed framework, SmartConSecurity implements three core workflows: Store, Retrieve, and Watchdog. These workflows define how data is securely stored, verified, and monitored throughout its lifecycle.

1) Store Logic

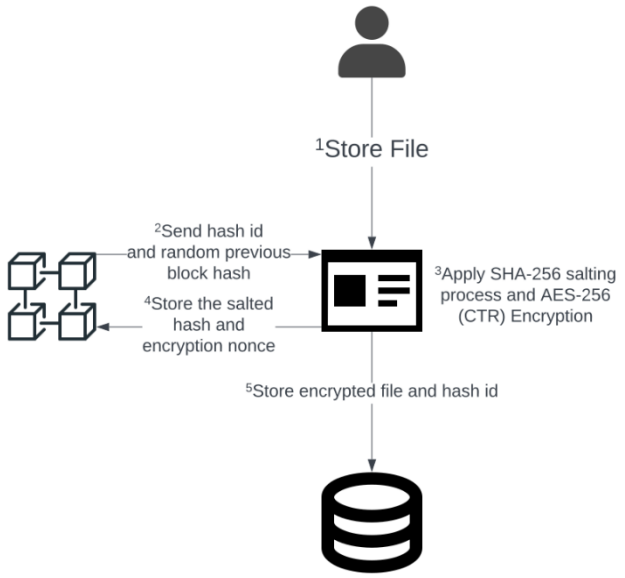

Figure 2. Store Workflow Logic

As shown in Figure 2, the storage workflow begins with user file submission, followed by the generation of a unique hash ID and a session-specific salt derived from blockchain entropy. The file is hashed using salted SHA-256, and the resulting hash is used as the AES-256-CTR encryption key to produce the ciphertext. The salted hash and encryption nonce are immutably recorded on the blockchain, while the encrypted file is stored off-chain in the local database. Pseudocode 1 formalizes the sequence of operations illustrated in Figure 2.

Notation:

- fd: Input file data
- fdHex: Hexadecimal representation of file data
- ma: MetaMask address of file owner
- hID: Unique file hash identifier (blockchain)
- fID: Unique file identifier (local database)
- randPrevBH: Random previous block hash (entropy)
- sh: Salted SHA-256 hash
- ct: Ciphertext
- nonce: Encryption nonce
- fEvent: Event payload on file modification

Pseudocode 1: Store Data

Input: fd, ma
Output: hID, sh, ct, nonce
1: Convert fd to hexadecimal:
fdHex ← BinToHex(fd)
2: Retrieve hash ID and random previous block hash:
hID ← GenID(ma)
randPrevBH ← GetRandPrevBlockHash()
3: Generate salted hash:
sh ← SHA256(fdHex + randPrevBH)
4: Encrypt data:
ct, nonce ← AES-256-CTR(fdHex, sh)
5: Commit verification data to blockchain:
SaveToBlockchain(sh, nonce)
6: Commit the encrypted file to the database
SaveToDatabase(hID, ct)

2) Retrieve Logic

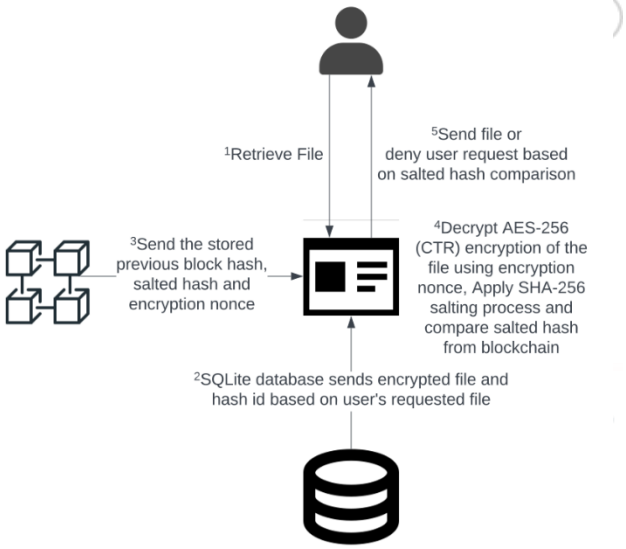

Figure 3. Retrieve Workflow Logic

As illustrated in Figure 3, the retrieval workflow begins when a user attempts to retrieve a file through the web application. Upon this request, the system retrieves the corresponding encrypted file and hash ID from the local database, while the associated salted hash, blockchain-derived salt, and encryption nonce are fetched from the blockchain. The ciphertext is then decrypted using AES-256-CTR, after which the salted SHA-256 hash is recomputed and compared against the on-chain record. The file is returned to the user only if the verification succeeds; otherwise, the request is denied. Pseudocode 2 formalizes the retrieval and verification process depicted in Figure 3.

Pseudocode 2: Retrieve Data

Input: fID, ma
Output: fd, errorMsg

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 10, Issue 2, February 2023

1: Retrieve encrypted file data and hash ID from the database:
ct, hID ← GetFileData(fID)
2: Retrieve salted hash, previous block hash, and encryption nonce from the blockchain:
sh, randPrevBH, nonce ← GetBCData(hID, ma)
3: Decrypt data:
fdHex ← AES-256-CTR-Decrypt(ct, sh, nonce)
4: Recompute salted hash for verification:
recompSH ← SHA256(fdHex + randPrevBH)
5: Compare hashes:
If recompSH != sh:
Return errorMsg
Else:
fd ← HexToBin(fdHex)
Return fd

3) Watchdog Logic

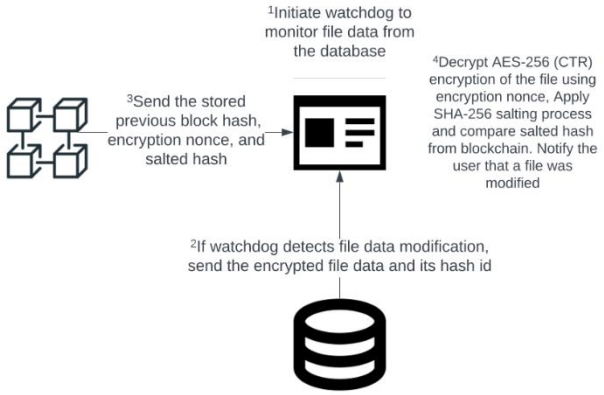

Figure 4. Watchdog Workflow Logic

As shown in Figure 4, the watchdog operates as an event-driven process that continuously monitors encrypted files stored in the local database. When a file modification is detected, the watchdog retrieves the altered ciphertext and its hash ID, then obtains the corresponding salted hash, blockchain-derived salt, and encryption nonce from the blockchain. The ciphertext is decrypted using AES-256-CTR, and the salted SHA-256 hash is recomputed and compared against the on-chain record. If a mismatch is detected, the system immediately flags the modification as unauthorized and notifies the user. Pseudocode 3 formalizes the monitoring and verification process illustrated in Figure 4.

Pseudocode 3: Watchdog

Input: fEvent
Output: alert
1: On file modification event:
ct, hID, fID, ma ← GetData(fEvent)
2: Retrieve verification data from blockchain:
sh, randPrevBH, nonce ← GetBCData(hID, ma)
3: Decrypt modified ciphertext:
fdHex ← AES-256-CTR-Decrypt(ct, sh, nonce)
4: Recompute salted hash for verification:
recompSH ← SHA256(fdHex + randPrevBH)
5: Compare recomputed hash with on-chain value:
If recompSH != sh:
alert ← NotifyUser()
return alert

IV. RESULTS AND DISCUSSION

A. Hamming Distance

The uniqueness of the hashing algorithm was evaluated by comparing SHA-256 digests with and without salt. For each test file, differing bit positions between the two digests were computed using Hamming distance. Since SHA-256 outputs 256 bits, the results were normalized and expressed as percentages, providing a clear measure of how much the hash output changed on average.

Table 1: Hamming Distance Test Results

File Name	Hamming Distance (%)
1-MB-DOC.doc	52
160-KB.txt	46
Free_Test_Data_1MB_JPG.jpg	50
Free_Test_Data_1OMB_MP3.mp3	51
Free_Test_Data_10.5MB_PDF.pdf	48
Free_Test_Data_15MB_MP4.mp4	52
RandomData.csv	48
Sample_HTML_for_testing.html	49
test_compressed_file.7z	44
test_gif.gif	42

As seen in Table 1, Testing achieved an average Hamming distance of approximately 48.2% across diverse file types, indicating that roughly half of the bits differ between original and salted hashes. Individual results ranged from 42% to 52%, demonstrating that salt addition produces unique and unpredictable hash outputs for all files regardless of format or content type. These results confirm that the hashing algorithm maintains strong resistance to correlation attacks and ensures session-based hash uniqueness.

B. Randomness Tests

Statistical randomness was evaluated using NIST-standard Frequency tests via Dataplot, with randomness accepted for p-values ≥ 0.01.

Table 2: Frequency (Monobit) Test Results

File Name	p-value	Result
1-MB-DOC.doc	0.532	Random
160-KB.txt	1.000	Random
Free_Test_Data_1MB_JPG.jpg	0.532	Random
Free_Test_Data_1OMB_MP3.mp3	0.708	Random
Free_Test_Data_10.5MB_PDF.pdf	0.382	Random
Free_Test_Data_15MB_MP4.mp4	0.382	Random
RandomData.csv	0.080	Random
Sample_HTML_for_testing.html	1.000	Random

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 10, Issue 2, February 2023

File Name	p-value	Result
test_compressed_file.7z	0.617	Random
test_gif.gif	0.453	Random

Table 3: Frequency (Block) Test Results

File Name	p-value	Result
1-MB-DOC.doc	0.320	Random
160-KB.txt	0.969	Random
Free_Test_Data_1MB_JPG.jpg	0.320	Random
Free_Test_Data_1OMB_MP3.mp3	0.925	Random
Free_Test_Data_10.5MB_PDF.pdf	0.561	Random
Free_Test_Data_15MB_MP4.mp4	0.561	Random
RandomData.csv	0.210	Random
Sample_HTML_for_testing.html	0.755	Random
test_compressed_file.7z	0.535	Random
test_gif.gif	0.570	Random

The Frequency (Monobit) Test evaluates whether the number of ones and zeros in the bit sequence is approximately equal, while the Frequency (Block) Test extends this analysis to subdivided blocks. In both tests, the null hypothesis of randomness is accepted when the p-value is greater than or equal to 0.01.

As shown in Tables 2 and 3, all test files produced p-values within the acceptable range, indicating statistically random distributions of bits. For the Monobit test, p-values ranged from 0.080 to 1.000, while for the Block test, they ranged from 0.210 to 0.969. Results confirm strong randomness meeting NIST criteria for cryptographic robustness. This further strengthens confidence that the hashing process does not introduce detectable patterns.

C. Collision Probability

The collision probability was estimated using the birthday paradox approximation $P \approx n^2 / 2^{257}$ Where n is the number of records. The denominator 2^{257} arises from the birthday bound calculation over the 2^{256}-sized SHA-256 output space, with the extra factor of 2 accounting for collision pair counting. This mathematical model reflects the likelihood that two distinct inputs will produce the same 256-bit hash value. Since the SHA-256 hash function operates over an extremely large space, the probability of collisions remains negligible even at very large dataset sizes, providing strong assurance of the algorithm's reliability for integrity verification.

Table 4: Estimated Collision Probability for Salted SHA-256 Hashes

Number of Records (n)	Example Scale	Collision Probability
103	Small DB (e.g., local system)	4.32×10−72
106	Medium DB (e.g., university)	4.32×10−66
109	Enterprise DB (e.g., banking)	4.32×10−60
1012	Massive DB (e.g., cloud-scale)	4.32×10−54

As shown in Table 4, even for small databases with 10^3 records, the probability of collision is approximately 4.32×10^{-72}. At medium to enterprise scales (10^6–10^9 records), the probabilities remain negligible, ranging from 10^{-66} to 10^{-60}. Even at massive cloud-scale systems with 10^{12} records, the chance of collision is only about 4.32×10^{-54}.

These results confirm that salted SHA-256 maintains strong collision resistance across both small to massive dataset sizes. This ensures that the SmartConSecurity framework can be confidently deployed in systems ranging from small institutional databases to large-scale enterprise and cloud environments.

D. System Performance Evaluation

The performance evaluation of the proposed system focuses on measuring its efficiency in handling file integrity and security operations, particularly the processes of salted hashing and encryption, as well as decryption. Since these operations are central to ensuring data confidentiality and tamper resistance within the blockchain-integrated architecture, it is essential to assess their computational cost and scalability across different file sizes.

For testing purposes, all input files were converted into their hexadecimal representation prior to hashing and encryption. In practical deployment, more efficient encodings may be applied; however, the hexadecimal format was selected in this evaluation to better illustrate differences in ciphertext representation, allowing the researchers to observe how the ciphertext changes within the database.

To provide a fair and end-to-end measurement, both encryption and decryption times were recorded, including the binary-to-hexadecimal conversion step.

Measuring Pipeline

- ***Encryption pipeline:*** File Input → Binary to Hexstring Conversion → Salted SHA-256 + AES-256-CTR → Ciphertext
- ***Decryption pipeline:*** Ciphertext → Salted SHA-256 + AES-256-CTR → Hexstring to Binary Conversion → File Output

Furthermore, the evaluation deliberately avoided the use of chunking techniques. While chunking can improve throughput for large-scale files, the objective of this study was to preserve end-to-end client–institution transparency, where a single file is treated as an indivisible unit of verification.

Test Setup

- ***Environment:*** React.js for front-end, FastAPI (Python) for backend.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 10, Issue 2, February 2023

- ***Input files:*** 5 MB, 50 MB, and 100 MB test datasets.
- ***Metrics:*** Average encryption time (s), average decryption time (s).
- *Procedure:* Each test was repeated three times, and the average value was reported.

Table 5: Encryption and Decryption Time Results

File Size	Hex Size	Avg. Encryption Time (s)	Avg. Decryption Time (s)
5 MB	10.5 MB	1.48	1.30
50 MB	105 MB	9.46	8.05
100 MB	210 MB	31.31	31.70

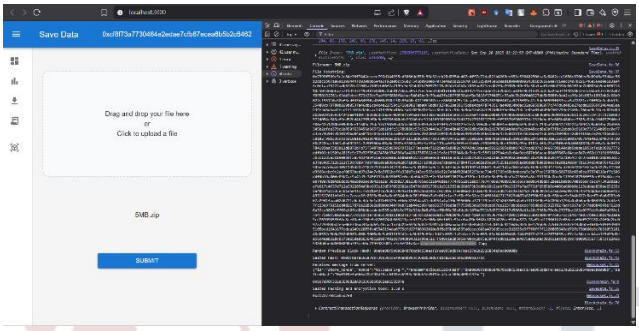

Figure 5. Lower Bound Encryption Performance (5 MB file)

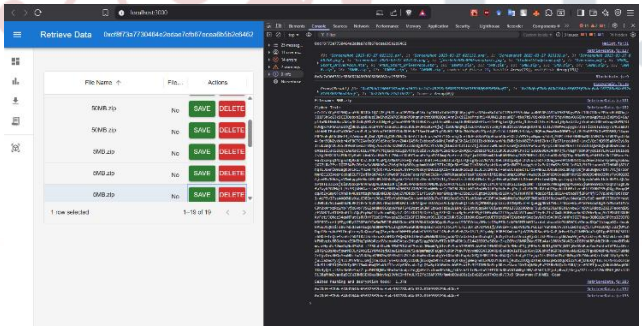

Figure 6. Lower Bound Decryption Performance (5 MB file)

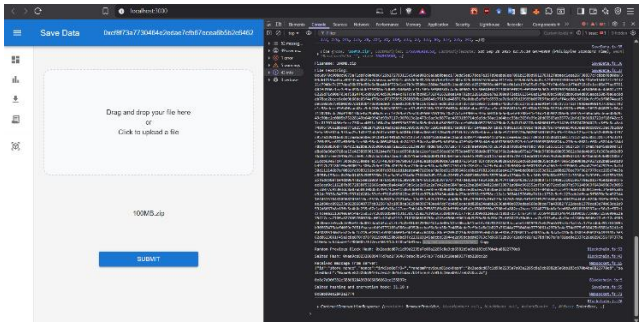

Figure 7. Upper Bound Encryption Performance (100 MB file)

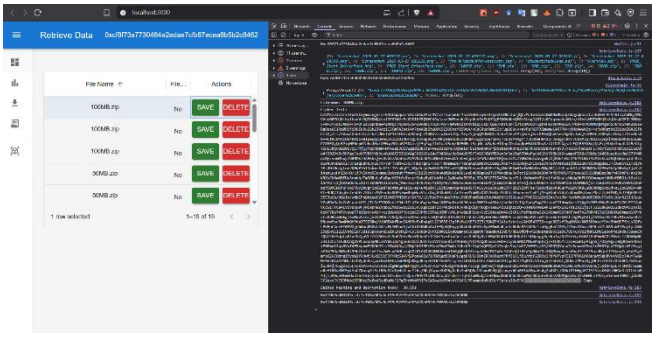

Figure 8. Upper Bound Decryption Performance (100 MB file)

As seen in Table 5, the results indicate that the hexadecimal conversion introduces a substantial portion of processing overhead, particularly for larger files. For the lower bound test using a 5 MB input file, the encryption and decryption, as shown in Figures 5 and 6, respectively, took 1.50s for encryption and 1.37s for decryption. For the upper bound evaluation using the 100 MB file, the encryption and decryption performance, as illustrated in Figures 7 and 8, took approximately 31.10s for encryption and 30.85 for decryption. The increasing overhead is largely attributed to the doubling of file size when converting binary data into hexadecimal format, resulting in significantly more data for both hashing and encryption pipelines to process. For example, the 100 MB file, when converted to hex (~210 MB), required over 31 seconds for both encryption and decryption. Despite this overhead, the system remains stable up to 100 MB without chunking.

E. Session-Based Salting Validation

To validate the effectiveness of session-based salting in preventing duplicate ciphertexts and file hash, identical files were uploaded multiple times across different sessions.

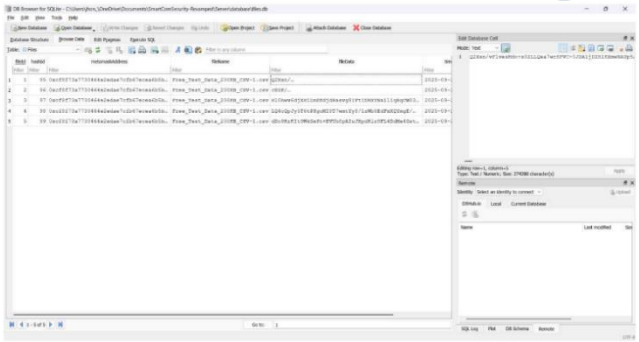

Figure 9. Ciphertext Variation for Identical Files

The validation involved uploading the same test file "Free_Test_Data_200KB_CSV-1.csv" 5 times in separate sessions. As demonstrated in Figure 9, the identical input file generated completely different cryptographic outputs.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 10, Issue 2, February 2023

Table 6: Session-Based Salting Detailed Summary

File Number	Hash ID	Random Previous Block Hash (Truncated)	Salted Hash (Truncated)	Encryption Nonce	Ciphertext (Truncated)
1	95	0xda23f8e7…	0xf129bcc1…	3AbWddmcpHQ=	Q2Xxn/vFlvesMdo+...
2	96	0x6660f942…	0xc8a8c4ae…	sH9jMc8efus=	cEGP/ibIDu84gKf+...
3	97	0xdb5a4fde…	0xdd5f0aa7…	rj+FcxliP2A=	Hl0hwv6djXSiLmEK...
4	98	0xbb29ab5b…	0x28201f63…	S8bAFUuPYlc=	LQ4cQpJy5T6tPSgo...
5	99	0xc0e41117…	0xfe59780d…	KqtAorVen9M=	dDo9RrKItOWkGeSt...

This validation confirms that SmartConSecurity's session-based salting mechanism successfully prevents ciphertext duplication and strengthens tamper resistance by ensuring each transaction produces unique cryptographic fingerprints, even for identical inputs, as demonstrated in Table 6.

F. Blockchain Integration Validation

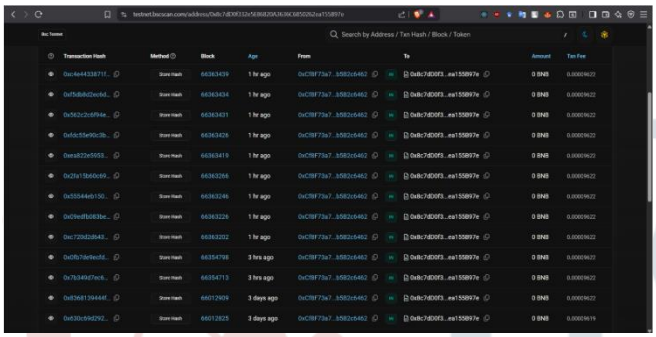

Figure 10. BNB Smart Chain Testnet Explorer Transaction History Interface

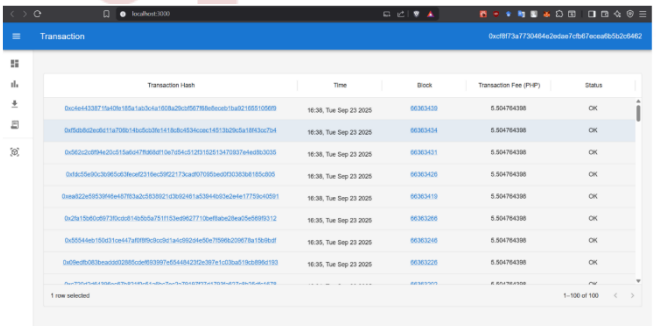

Figure 11. Blockchain Transaction History Interface

The SmartConSecurity framework was implemented and tested on the BSC through a custom smart contract to validate blockchain integration in a practical setting. The blockchain anchoring was confirmed using the BNB Smart Chain Testnet Explorer, which recorded transactions associated with the Store Hash method as seen in Figure 10. Each transaction entry displays the corresponding transaction hash, method, block number, timestamp, and transaction fee, confirming that the data was successfully written to the blockchain and is publicly verifiable.

In parallel, the system's web interface provided a real-time view of the same blockchain events as seen in Figure 11. The interface retrieved transaction details directly from the BSC network and displayed key attributes such as transaction hash, block height, timestamp, and fee. Each operation was marked with a status of OK, confirming successful anchoring and consistency with the explorer's records.

G. Watchdog System Validation

The real-time file monitoring capability was validated by simulating unauthorized database modifications and confirming that the watchdog system successfully detected and raised an alert. While its response times were not formally benchmarked in this prototype stage, the system consistently identified tampering events immediately due to the limited dataset size.

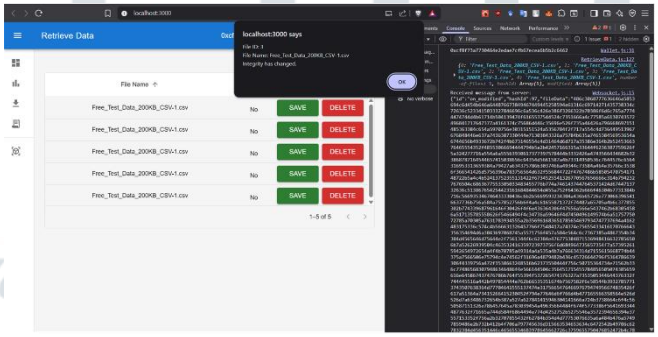

Figure 12. Watchdog System Notification on File Integrity Change

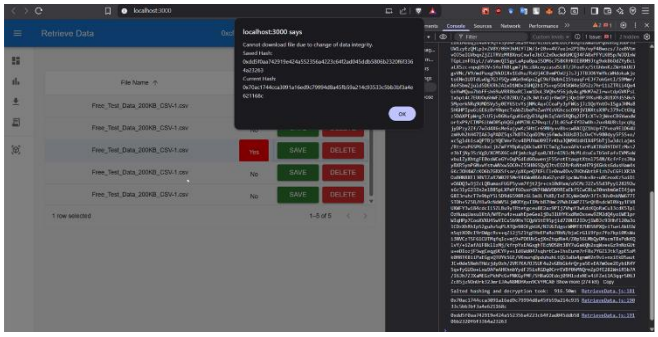

Figure 13. Watchdog System Notification on File Integrity Change

As shown in Figure 12, when a file's data was modified, the system automatically initiated an integrity verification process. The affected file was then tagged as modified, and as seen in Figure 13, the watchdog restricted user access to prevent the retrieval of potentially malicious or corrupted

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 01, January 2026

content.

V. CONCLUSION AND RECOMMENDATIONS

This study presents SmartConSecurity, a hybrid database system that addresses the institutional trust deficit in data management by enabling client-controlled integrity verification. The system combines SQLite local storage with salted SHA-256 hashing, AES-256-CTR encryption, and blockchain-based verification to provide transparent, trustless data stewardship.

Unlike blockchain-based provenance and integrity frameworks that emphasize traceability or post-hoc verification [13], SmartConSecurity unifies encryption, integrity verification, and real-time monitoring within a lightweight database-centric workflow. While public blockchains provide strong immutability guarantees, their transparency introduces potential exposure risks [14], motivating SmartConSecurity's focus on local enforcement over cloud-centric storage models [15]. Future extensions may incorporate domain-specific integrity authentication techniques demonstrated in blockchain-integrated systems [16]. Compared to tamper-resistant audit logging approaches that support post-incident analysis [17], [18], SmartConSecurity enables immediate detection of unauthorized modifications through an event-driven watchdog.

The framework's cryptographic validation demonstrates robust security properties: an average Hamming distance of approximately 48.2%, statistical randomness (p-values ≥ 0.01), and negligible collision probability ($<10^{-54}$ for 10^{12} records). Performance evaluation further confirms the system's practicality.

End-to-end measurements, including binary-to-hexadecimal conversion, indicate that SmartConSecurity efficiently handles files up to 100 MB, demonstrating that the system maintains end-to-end encryption and decryption within practical time bounds, even when including hexadecimal conversion, highlighting both security and operational feasibility.

The session-based salting mechanism ensured differentiation across identical file uploads, with each transaction generating unique salted hashes, encryption nonces, and ciphertexts. This effectively prevents replay attacks and guarantees dynamic verification integrity. Blockchain integration, validated through deployment on the BSC testnet, confirmed successful immutable anchoring of verification data, with transaction records independently verifiable through blockchain explorers. The watchdog system further demonstrated real-time tampering detection by immediately identifying unauthorized modifications and restricting access to potentially compromised files.

SmartConSecurity establishes a foundation for transforming institutional transparency. By enabling client-controlled verification, the system transforms institutional data management from a trust-based model to a verify-based model, where transparency becomes technologically guaranteed rather than organizationally promised. The system enables clients to independently verify how their data is managed without requiring trust in institutional claims. By separating data storage from integrity verification across different technological domains, institutions can demonstrate their data stewardship practices rather than merely assert them.

While SmartConSecurity demonstrates proof of concept in a controlled environment, real-world deployment introduces challenges related to scalability, integration, operational efficiency, and user trust. Future work will address these by deploying a private blockchain using Hyperledger Fabric, migrating to a server-based RDBMS for concurrent data access, and benchmarking under enterprise workloads to evaluate latency and transaction throughput.

Future work should prioritize:

1) ***Private Blockchain Deployment (Hyperledger Fabric)*** – Reducing operational costs while maintaining immutability guarantees.
2) ***Performance Optimization*** – Comprehensive benchmarking of system overhead under enterprise workloads.
3) ***Database Scalability Enhancement*** – Transition from SQLite to server-based RDBMS (PostgreSQL, MySQL) to enable multi-user concurrency, transaction isolation, and enterprise-scale data management.
4) ***Broader Industry Applications*** – Adapt the framework for domains that require high trust and verifiable transparency, ensuring its applicability across institutional and enterprise environments.

SmartConSecurity provides a practical framework for institutions to move beyond opacity toward verifiable transparency, offering clients the technological means to validate data integrity claims independently. This approach has significant implications for sectors that require high trust assurance, including, but not limited to, healthcare, finance, government, and education.

REFERENCES

[1] J. Clarke, Sql injection attacks and defense. Syngress Media,U.S, 2012.

[2] A. A. Onyekachi and D. O. Njoku, "SQL Injection Attack on Web Base Application: Vulnerability Assessments and Detection Technique," ResearchGate, vol. 08, no. 03, Jul. 2021, [Online]. Available: https://www.researchgate.net/publication/353257660_SQL_Injection_Attack_on_Web_Base_Application_Vulnerability_Assessments_and_Detection_Technique

[3] A. Mousa, M. Karabatak, and T. Mustafa, "Database Security Threats and Challenges," 2020 8th International Symposium on Digital Forensics and Security (ISDFS), Jun. 2020, doi: https://doi.org/10.1109/isdfs49300.2020.9116436.

[4] D. Rachmawati, J. T. Tarigan, and A. B. C. Ginting, "A comparative study of Message Digest 5(MD5) and SHA256 algorithm," Journal of Physics: Conference Series vol. 978, Mar. 2018, doi: https://doi.org/10.1088/1742-6596/978/1/012116.

[5] S. S. Srivastava, Medha Atre, S. Sharma, R. Gupta, and S. K. Shukla, "Verity: Blockchains to Detect Insider Attacks in DBMS," arXiv (Cornell University), Jan. 2019, doi: https://doi.org/10.48550/arxiv.1901.00228.

[6] B. Gipp, Jagrut Kosti, and C. Breitinger, "Securing Video Integrity Using Decentralized Trusted Timestamping on the Bitcoin Blockchain," AIS Electronic Library (AISeL), 2016. https://aisel.aisnet.org/mcis2016/51

[7] E. O. Kiktenko, M. A. Kudinov, and A. K. Fedorov, "Detecting BruteForce Attacks on Cryptocurrency Wallets," Business Information Systems Workshops, pp. 232–242, Dec. 2019, doi: https://doi.org/10.1007/978-3-030-36691-9_20

[8] M. A. Almaiah, L. M. Saqr, L. A. Al-Rawwash, L. A. Altellawi, R. AlAli, and O. Almomani, "Classification of Cybersecurity Threats, Vulnerabilities and Countermeasures in Database Systems," Computers, Materials and Continua, vol. 81, no. 2, pp. 3189–3220, Nov. 2024, doi: https://doi.org/10.32604/cmc.2024.057673.

[9] H. Ahmed, "A Review of Hash Function Types and their Applications," Wasit Journal of Computer and Mathematics Science, vol. 1, no. 3, pp. 120–139, Oct. 2022, doi: https://doi.org/10.31185/wjcm.52.

[10] P. Natho, S. Somsuphaprungyos, S. Boonmee, and S. Boonying, "Comparative study of password storing using hash function with MD5, SHA1, SHA2, and SHA3 algorithm," International Journal of Reconfigurable and Embedded Systems (IJRES), vol. 13, no. 3, p. 502, Nov. 2024, doi: https://doi.org/10.11591/ijres.v13.i3.pp502-511.

[11] Z. Chen, J. Gu, and H. Yan, "HAE: A hybrid cryptographic algorithm for blockchain medical scenario applications," Applied Sciences, vol. 13, no. 22, Art. no. 12163, Nov. 2023, doi: https://doi.org/10.3390/app132212163.

[12] R. Kalis and A. Belloum, "Validating Data Integrity with Blockchain," 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Dec. 2018, doi: https://doi.org/10.1109/cloudcom2018.2018.00060.

[13] U. Javaid, M. N. Aman, and B. Sikdar, "BlockPro," Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems - BlockSys'18, 2018, doi: https://doi.org/10.1145/3282278.3282281.

[14] H. Byun, J. Kim, Y. Jeong, B. Seok, S. Gong, and C. Lee, "A Security Analysis of Cryptocurrency Wallets against Password Brute-Force Attacks," Electronics, vol. 13, no. 13, pp. 2433–2433, Jun. 2024, doi: https://doi.org/10.3390/electronics13132433.

[15] H. Bhalla, "Blockchain-Based Framework for Secure Cloud Storage with Data Integrity Verification," Master's thesis, National College of Ireland, Dublin, 2024.

[16] D. Xu, N. Ren, and C. Zhu, "Integrity Authentication Based on Blockchain and Perceptual Hash for Remote-Sensing Imagery," Remote Sensing, vol. 15, no. 19, p. 4860, Jan. 2023, doi: https://doi.org/10.3390/rs15194860.

[17] M. B. Thazhath, J. Michalak, and T. Hoang, "Harpocrates: Privacypreserving and immutable audit log for sensitive data operations," in Proc. IEEE Int. Conf. Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), Dec. 2022, doi: https//doi.org10.1109/TPS-ISA56441.2022.00036.

[18] T. H. Austin and F. D. Troia, "A Blockchain-Based Tamper-Resistant Logging Framework," Communications in Computer and Information Science, pp. 90–104, Jan. 2023, doi: https://doi.org/10.1007/978-3-031- 24049-2_6.