The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 167-176
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9999

Digital Evidence Integrity System Using SHA-256 Hashing for Digital Forensic Data Tracking

Yuda Perwira*, Humala Simangunsong, Yogi Irwan Syahputra, Selvi Yolanda
Informatics Engineering Study Program, STMIK Pelita Nusantara, Medan, Indonesia
Email: [1,*]yudaperwira25@gmail.com, [2]humala99@gmail.com, [3]yogiirwansyahputra@gmail.com, [4]selviyolanda71@gmail.com
(* : yudaperwira25@gmail.com)
Submitted: **29/06/2026**; Accepted: **20/07/2026**; Published: **29/07/2026**

Abstract—This study presents the design and implementation of a digital forensic system that integrates an audit trail mechanism with the SHA-256 hashing algorithm to track data changes and detect unauthorized manipulation, without depending on a blockchain architecture. Using a Design Science Research approach, the system called the Digital Evidence Integrity System was built as a web application with PHP and MySQL, covering requirement identification, model design, implementation, an integrity validation mechanism, and testing and evaluation. Evaluation began with a pilot of five records under authorized and unauthorized update scenarios that simulated direct database access, then was expanded to sixteen scenarios spanning seven manipulation categories: insert, update, deletion, replay, concurrent access, stored-hash alteration, and audit-trail-log manipulation. Across the fourteen scenarios usable for a confusion matrix, the system correctly flagged 4 of 9 genuine manipulation attempts (a 44.4% detection rate) with zero false positives on legitimate changes, while consistently failing to detect four categories of manipulation: outright deletion of a record, replay of a superseded but internally valid data-hash pair, an attacker recomputing and overwriting the stored hash together with the data, and tampering with the audit-trail log itself. A repeated one-character perturbation test (60 trials) confirmed the avalanche effect of SHA-256, with a mean Hamming distance of 126.42 of 256 bits (49.38%, SD 8.52) between the hash of the original and the perturbed input. Taken together, these findings indicate that an audit trail combined with SHA-256 hashing is computationally lightweight and reliably detects manipulation that alters data without also updating its stored hash, but is not, in its current form, a sufficient safeguard against an attacker capable of also controlling the hash or the log; protecting the audit-trail log itself, adding a genuine user-authentication mechanism, and closing the deletion, replay, and stored-hash-alteration gaps identified here are the priorities for further development before the system could be relied upon in an operational forensic setting.

Keywords: Digital Forensics; Data Integrity; Audit Trail; SHA-256; Hashing

1. INTRODUCTION

The rapid advancement of information technology has made digital data an increasingly central component of many systems, including its use as evidence in legal proceedings. In Indonesia, the admissibility of electronic evidence is regulated under Article 5 of Law No. 11 of 2008 on Electronic Information and Transactions (UU ITE), as amended by Law No. 19 of 2016 and, most recently, by Law No. 1 of 2024 on the Second Amendment to Law No. 11 of 2008, which affirms that electronic information and/or electronic documents, and/or their printouts, constitute valid legal evidence. A major challenge nonetheless remains: the integrity and authenticity of digital data are not yet fully guaranteed, since digital data can easily be altered, copied, or manipulated without leaving any visible trace on the object itself.

Data integrity plays a decisive role in determining whether a piece of electronic evidence can be admitted before a court, because a judge or investigator must be able to demonstrate that the presented data is identical to the data originally acquired [1], [2]. On the other hand, hashing techniques have long been used in digital forensics to help ensure this integrity. Hashing produces a unique digital fingerprint of a set of data, one that changes significantly whenever even a small part of the underlying data is modified [3], [4], [5], [6], [7]. This property makes hashing algorithms, such as SHA-256, a practical building block for systems that need to verify whether data has remained unchanged over time.

Several prior studies have explored blockchain as a means of securing digital evidence and its chain of custody [8], [9], [10], [11], [12], [7], [13]. While blockchain offers strong guarantees through distributed consensus, it is also comparatively complex to implement and resource-intensive to operate, which can be excessive for small- to medium-scale information systems that do not require distributed trust across multiple independent parties. Most current information systems, meanwhile, have not integrated any cryptography-based audit trail mechanism capable of fully guaranteeing data integrity on their own. This reveals a gap between the practical need for a reliable, lightweight evidentiary mechanism and the technology that is readily available to small institutions and research teams.

Based on this gap, the present study focuses on developing a hashing-based digital forensic system, referred to as the Digital Evidence Integrity System, which systematically tracks data changes and verifies their integrity through the combination of an audit trail module and the SHA-256 hashing algorithm. Unlike the blockchain-based approaches used in previous work, the proposed system adopts a centralized audit-trail-with-hashing design that is simpler to implement, lighter in computational resources, and therefore more readily deployable in small-scale institutional settings, while still providing an initial foundation toward a more complete digital chain-of-custody mechanism in future development.

The novelty of this study lies in three aspects: (1) the development of an integrated, hash-based audit trail rather than conventional logging alone; (2) a continuous data-change tracking approach capable of directly detecting manipulation through hash comparison; and (3) the provision of an initial foundation toward a digital chain of custody without dependence on a complex blockchain architecture. Thus, this study does not merely use hashing as a validation tool, but develops it into an integrated audit-trail-and-hashing prototype capable of detecting integrity mismatches and

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 167-176
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9999

supporting data traceability. It should be noted that the prototype currently demonstrates integrity-mismatch detection only; it does not yet provide authenticated users, protected logs, acquisition provenance, evidence-handling procedures, or a complete chain-of-custody process, and should therefore be understood as an initial building block toward a complete digital forensic system rather than as such a system in itself.

Based on the background above, this study addresses three problem statements: (1) how a digital-forensic-based data-change tracking system can be designed; (2) how the SHA-256 hashing algorithm can be implemented to ensure data integrity; and (3) how effectively the system can detect data manipulation based on changes in hash values. Correspondingly, the objectives of this study are to design a digital-forensic-based data-change tracking system capable of systematically recording every change activity, to implement the SHA-256 hashing algorithm within the system to ensure the integrity of stored data, and to test and evaluate the system's ability to detect data manipulation by comparing hash values before and after a change.

The remainder of this article is organized as follows. Section 2 describes the research methodology, including the Design Science Research approach and the stages followed in developing the system. Section 3 presents the results of system implementation and testing, together with a discussion of the findings and the limitations identified during evaluation. Section 4 concludes the article and outlines directions for future development.

It should be emphasized that the contribution of this study is not the hashing algorithm itself, since SHA-256 is a well-established cryptographic primitive, but rather the way the algorithm is engineered into a coherent audit-trail architecture that a small institution can realistically deploy and maintain. By keeping the design centralized rather than distributed, the system trades some of the tamper-resistance guarantees of blockchain for substantially lower implementation cost, faster response time, and simpler maintenance, a trade-off that is made explicit and evaluated later in Section 3.

2. RESEARCH METHODOLOGY

2.1 Research Approach

This study employs a Design Science Research (DSR) approach, following the process model of Peffers et al [14]. which focuses on developing an artifact, in this case a digital forensic system, to address the problems of data integrity and change traceability described in Section 1. DSR was selected because it structurally combines the aspects of design, implementation, and system evaluation, allowing the resulting artifact to be assessed both for its technical correctness and its practical usefulness. The five research stages carried out in this study (Table 1) map onto the six generic DSR activities as follows: problem and requirement identification corresponds to problem identification and motivation together with the definition of objectives; system model design corresponds to design and development; system implementation corresponds to the practical construction of the artifact within design and development; the integrity validation mechanism and system testing correspond to demonstration and evaluation; and the resulting analysis and article preparation correspond to communication.

Within this approach, the object being engineered, the Digital Evidence Integrity System, is treated simultaneously as a research artifact and as a working prototype: its internal logic must be defensible on cryptographic grounds, while its interface and workflow must remain usable by non-specialist administrative staff who are the intended end users in a small institutional setting such as STMIK Pelita Nusantara.

2.2 Research Stages

The problem-solving strategy was carried out through five stages, summarized in Table 1 together with the main activity and achievement indicator of each stage; the person in charge of each stage was tracked internally in the project's research management plan and is not reproduced in Table 1 for conciseness.

a. Problem and requirement identification. The initial stage involved analyzing the weaknesses of existing information systems, particularly the absence of an audit trail mechanism capable of ensuring data integrity, and identifying system requirements covering data-change logging, integrity validation, and manipulation-detection capability.
b. System model design. Based on the identified requirements, a system model was designed consisting of a data-change logging module (audit trail), a hashing module to generate a digital fingerprint, and an integrated log storage structure. Every stored record is assigned a hash value generated with the SHA-256 algorithm, so that any change to the data can be detected through a change in the hash value.
c. System implementation. The designed model was implemented as a web-based application using PHP 8.3 with a MySQL database (MariaDB 10.11), running in XAMPP 8.2.12. Development and testing were carried out on a machine running Windows 10 with an Intel Core i3 processor and 8 GB of DDR4 RAM. Database access during testing was restricted to a dedicated, non-root application account with a password (rather than the default root account without a password), and access to phpMyAdmin was restricted to localhost. The source code is available in a public repository at https://github.com/yudaperwira25-web/Digital-Evidence-Integrity-.git. The system records every data-change activity, stores the hash values before and after each change, and records the timestamp and the user involved.

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 167-176
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9999

d. Integrity validation mechanism. Validation is performed by comparing the hash value of the data before and after a change. If the two hash values differ, the system identifies that a change or manipulation has occurred, allowing data integrity to be verified objectively rather than through manual inspection. The threat model this mechanism is designed for is an actor who bypasses the application's own update workflow (for example, through a direct database query) but does not also modify the stored hash. Because the hash is unkeyed and stored in the same database as the protected record, an attacker with write access to that database can, in principle, recompute and overwrite the stored hash together with the data, leaving no detectable mismatch; the current design therefore does not defend against an attacker with that level of access. Closing this gap would require a keyed construction such as HMAC-SHA-256 with a secret held outside the database, a digital signature scheme, or writing the hash to an independently protected, append-only log rather than alongside the data it protects; these are noted as directions for future development rather than claims about the present prototype.
e. System testing and evaluation. Testing was carried out through simulated data-change scenarios comprising authorized data changes made through the application and unauthorized data changes made directly on the database. The system was evaluated on its ability to detect data changes, maintain audit-trail consistency, and ensure the accuracy of manipulation identification.

The test results were subsequently analyzed to assess the effectiveness of the approach, focusing on the reliability of the system in ensuring data integrity and its potential application within the broader context of digital forensics. Each stage in Table 1 was treated as a checkpoint: work did not proceed to the next stage until the corresponding achievement indicator had been met, which kept the five-stage plan aligned with the overall one-year timeline of the research grant that funded this study.

Table 1. Research Stages and Achievement Indicators

Research Stage	Main Activity	Achievement Indicator
1. Requirement Analysis	Literature study and system requirement analysis	Requirement document completed
2. System Design	Audit trail, hashing, and database (ERD) design	Complete system design document
3. Implementation	Web-based development with SHA-256 and audit trail	System runs according to design
4. System Testing	Functional and data-integrity testing	Manipulation detection ≥ 95%
5. Evaluation	Result analysis and article preparation	Article ready for submission

3. RESULTS AND DISCUSSION

3.1 Overview of the Developed System

The Digital Evidence Integrity System has been implemented as a web-based application using PHP 8.3 with a MySQL database (MariaDB 10.11), running in an XAMPP environment. The system consists of three main modules, as designed in Section 2: (1) the audit-trail logging module, (2) the SHA-256 hashing module, and (3) the log storage and validation module. The database consists of two main tables, data_identitas, which stores the protected data together with its latest hash value (current_hash), and audit_trail, which stores the chronological history of every data change. The main dashboard view of the system is shown in Fig. 1.

The data_identitas table stores three protected fields for each record, namely name, national identity number (NIK), and address, alongside the current_hash column that always reflects the hash value computed at the time of the last change made through the application. The audit_trail table is append-only from the application's point of view: every insert, update, and validation event creates a new row rather than modifying an existing one, so that the table itself forms a running history of the record's lifecycle. This separation between the protected data and its change history is what allows the validation module, described in Section 3.4, to compare what the data currently contains against what it should contain according to the last authorized change.

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 167-176
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9999

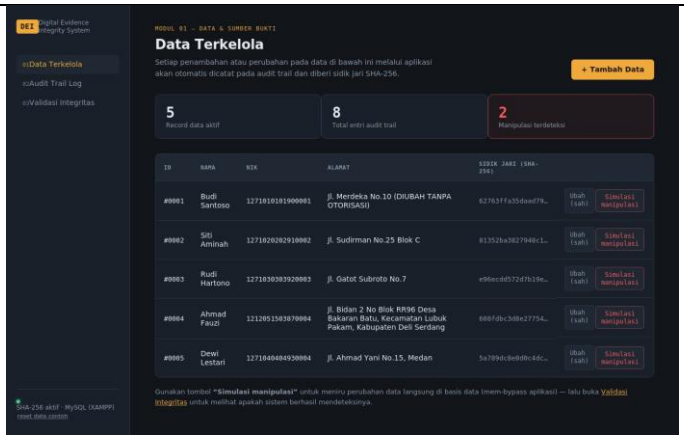

Fig 1. Managed Data Dashboard of the Digital Evidence Integrity System

3.2 SHA-256 Hashing Module

The hashing module generates a hash value from the data content, namely the name, NIK (national identity number), and address fields, using PHP's hash('sha256', $payload) function. The payload is canonicalized before hashing as a compact JSON object with the three field keys sorted alphabetically (alamat, nama, nik), each value taken as an ASCII/UTF-8 string with no added whitespace between the JSON delimiters; for example, the record used as the illustration below serializes to {"alamat":"Jl. Bidan 2 No Blok RR69 Desa Bakaran Batu, Kecamatan Lubuk Pakam, Kabupaten Deli Serdang","nama":"Ahmad Fauzi","nik":"1212051503870004"}, a 148-byte (1,184-bit) string. This guarantees a consistent hash output regardless of the order in which fields are supplied. The current implementation does not yet define an explicit policy for null or empty field values or for Unicode normalization (e.g., NFC/NFD) of non-ASCII input; this is noted as an implementation detail to be formalized in future work rather than a validated part of the present canonicalization procedure. Formally, the hash value H of a record is computed as shown in (1), where payload is the canonical JSON representation of the record.

Sorting the JSON keys before hashing is a deliberate design choice rather than an incidental detail. Because JSON does not guarantee a fixed key order, two payloads containing exactly the same field values could otherwise be serialized differently by different parts of the application, or by a different PHP version, producing two different hash values for what is semantically identical data. Canonicalizing the key order first removes this source of false mismatches, so that a hash difference can always be attributed to an actual change in the underlying data rather than to an artifact of serialization.

$$H = \text{SHA-256}(\text{payload}) \quad (1)$$

As an illustration, data with the name “Ahmad Fauzi” and the address “Jl. Bidan 2 No Blok RR69 Desa Bakaran Batu, Kecamatan Lubuk Pakam, Kabupaten Deli Serdang” produces the following 64-character hexadecimal hash value:

608fdbc3d8e277541c1fd0ca630b200eab7ce2e4cbeab96607e23bf137ac7ac1

When the block number in the address is altered from “RR69” to “RR96”, so that only two characters of the input change, the recomputed hash becomes:

3ca04eca56f267f28c6aa4cda859275de4f22960ab7f387854a01af21d54aac6

Comparing the two 256-bit outputs bit by bit, 106 of the 256 bits differ (a Hamming distance of 106, or 41.4% of the output), consistent with the avalanche effect of SHA-256, expressed conceptually in (2): a minimal change Δx in the input produces an output difference that is, on average, close to 50% of the output bits, denoted $H(x) \oplus H(x + \Delta x) \approx$ random. Because a single worked example does not by itself statistically validate this property, the one-character perturbation test was repeated across 60 independent trials, applying a single-character change at a random position in one of the three protected fields (name, NIK, or address) for three base records, using the same canonical JSON payload and hash('sha256', $payload) computation as the deployed system. Across these 60 trials, the Hamming distance had a mean of 126.42 bits (49.38% of the 256-bit output, close to the theoretical 50%), a median of 126 bits, a standard deviation of 8.52 bits, a range of 108 to 147 bits, and a 95% confidence interval of 124.26 to 128.57 bits for the mean; the result was consistent across all three perturbed fields (mean 125.65 bits for name, 126.63 for NIK, and 127.17 for address), indicating that the avalanche behavior is not specific to the single field illustrated above.

$$H(x) \oplus H(x + \Delta x) \approx \text{random}, \quad \text{for minimal } \Delta x \quad (2)$$

This sensitivity to even the smallest change is the cryptographic property underlying the system's manipulation-detection capability: any unauthorized alteration to a protected record, however small, will produce a hash value that no longer matches the value recorded at the time of the last authorized change.

3.3 Audit Trail Module

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 167-176
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9999

The audit-trail module automatically records every data-change activity, including the action type (INSERT, UPDATE, or VALIDATION_MISMATCH), the user associated with the change, the timestamp, and the hash values before and after the change. During the present testing, the recorded user identity was manually selected from a dropdown rather than derived from an authenticated login session; until a genuine authentication mechanism is implemented, the audit trail records who was designated as having made a change, not who is cryptographically verified to have made it. It is implemented through the log_audit() function in the system's functions.php file. Fig. 2 shows an example of the audit-trail log produced during testing, including the entry created automatically when a manipulation is detected.
Each log entry is deliberately kept immutable at the application layer: there is no update or delete operation exposed for the audit_trail table through the user interface, so that, under normal use through the application, the history of a record can only grow and never be edited retroactively. When the validation module later detects a mismatch, it does not overwrite the earlier INSERT or UPDATE entries; instead, it appends a new VALIDATION_MISMATCH entry, preserving the full sequence of what was recorded, when the mismatch was found, and what the two conflicting hash values were.

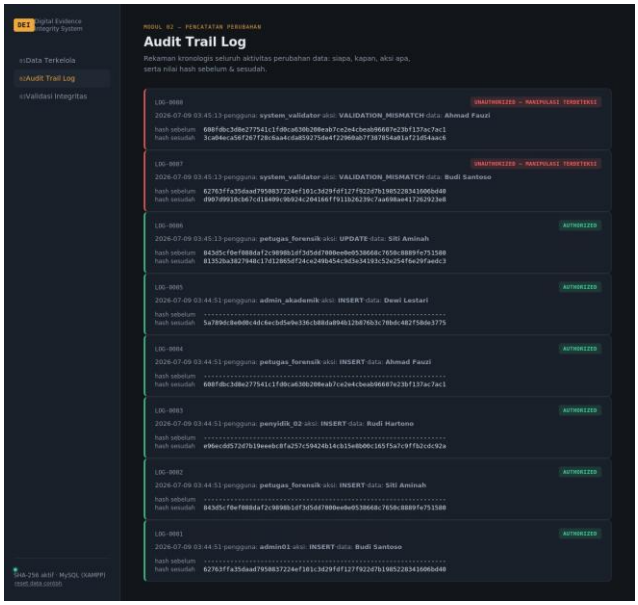

Fig 2. Audit Trail Log Displaying the History of Data Changes

3.4 Integrity Validation Module and Testing Results

The validation module recomputes the hash value from the data currently stored in the database and compares it with the current_hash value last recorded when the data was legitimately changed through the application. If the two values do not match, the system flags the data as manipulated and automatically records the finding as a new audit-trail entry with the status unauthorized. Fig. 3 shows the result display of the validation module.

Fig 3. Data Integrity Validation Module Result

Functional testing was performed on all modules in a local environment (XAMPP, PHP 8.3, MariaDB 10.11) to confirm that the system operates according to its design. The results, summarized in Table 2, show that all five tested functions, audit-trail logging, hash generation, mismatch detection, audit-trail display, and the manipulation simulation itself, operated successfully.

Table 2. System Functional Testing Results

Module	Function Tested	Status
Audit Trail	Data-change logging (INSERT/UPDATE)	Successful
Hashing	Generate SHA-256 hash from new data	Successful
Validation	Hash mismatch detection	Successful
Web Interface	Audit trail history display	Successful

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 167-176
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9999

Module	Function Tested	Status
Manipulation Simulation	Direct data change bypassing the application	Successful

Data integrity testing was initially conducted as a small pilot test on five records (SKN-01 to SKN-05), comprising one authorized change made through the application, two unauthorized changes simulating direct database access, and two records left unchanged as controls. Following this pilot, testing was expanded to eleven additional scenarios (SKN-06 to SKN-16) covering seven manipulation categories identified as necessary for a more representative evaluation: insert, update, deletion, replay, concurrent access, stored-hash alteration, and audit-trail-log manipulation. Each authorized action was performed through the application's real HTTP endpoints (the same code path a legitimate user would exercise), and each unauthorized action was performed through a direct SQL query against the database, bypassing the application entirely, after which the system's actual validation routine, not a manual or simulated check, was used to determine the outcome. The scenarios and results, with hash values truncated to the first 20 characters for readability, are summarized in Table 3.

The two unauthorized scenarios were deliberately constructed to represent different attacker capabilities. SKN-02 simulates a change made by triggering the application's own manipulation-simulation button, which writes directly to the data_identitas table while bypassing the normal update workflow and its hashing step. SKN-03 goes further and simulates a change made through a manual SQL query executed outside the application entirely, for example through phpMyAdmin, representing an attacker with direct database credentials rather than merely application access. Distinguishing these two cases was intended to confirm that detection does not depend on which entry point the attacker used, only on whether the recorded hash and the recomputed hash of the current data agree.

Table 3. Data Integrity Testing Results (SKN-01 to SKN-16, covering seven manipulation categories)

Code	Type of Change	Validation Result
SKN-01	Authorized change (UPDATE via application)	VERIFIED
SKN-02	Unauthorized change (button simulation)	MANIPULATION DETECTED
SKN-03	Unauthorized change (direct manual query to DB)	MANIPULATION DETECTED
SKN-04	No change (control)	VERIFIED
SKN-05	No change (control)	VERIFIED
SKN-06	Insert (authorized, via application)	VERIFIED
SKN-07	Insert (unauthorized, direct DB, hash left blank)	MANIPULATION DETECTED
SKN-08	Update (authorized, via application)	VERIFIED
SKN-09	Update (unauthorized, direct DB, hash not updated)	MANIPULATION DETECTED
SKN-10	Deletion (authorized) — not testable, no delete feature exists	N/A
SKN-11	Deletion (unauthorized, direct DB)	NOT DETECTED
SKN-12	Replay (superseded but valid data–hash pair restored via DB)	VERIFIED (false negative)
SKN-13	Concurrent update (two near-simultaneous authorized edits)	VERIFIED, but a lost update occurred
SKN-14	Stored-hash alteration (data and hash both recomputed by attacker)	VERIFIED (false negative)
SKN-15	Audit-log manipulation (deletion of log entries)	NOT DETECTED
SKN-16	Audit-log manipulation (edit of a log entry's status)	NOT DETECTED

Based on Table 3, the picture that emerges from the expanded test set is more nuanced than the pilot alone suggested. Of the seven manipulation categories tested, three were reliably detected: unauthorized updates that did not also alter the stored hash (SKN-02, SKN-03, SKN-09) and an unauthorized insert with no hash recorded (SKN-07). Four categories were not detected by the current mechanism: deletion of a record (SKN-11), replay of a superseded but internally consistent data-hash pair (SKN-12), stored-hash alteration in which the attacker recomputes and overwrites the hash together with the data (SKN-14), and manipulation of the audit-trail log itself, whether by deletion (SKN-15) or by editing a log entry's recorded status (SKN-16). Excluding SKN-10 (authorized deletion, not testable because the prototype has no delete feature) and SKN-13 (a concurrency/data-consistency finding rather than a manipulation-detection test, discussed separately below), the 14 remaining scenarios yield a confusion matrix of 4 true positives, 5 true negatives, 0 false positives, and 5 false negatives. This corresponds to a detection rate (recall) of 4/9, or 44.4%, among genuine manipulation attempts, a precision of 100% (no legitimate change was ever flagged as manipulation), and an overall accuracy of 9/14, or 64.3%, across the scenario set. This is substantially lower than the 100% figure obtained on the original five-record pilot, and this later, larger, and more adversarial test set is the one on which the system's detection capability should be judged: the mechanism reliably catches manipulation that changes the data without also updating the stored hash, but it does not, in its present form, catch an attacker capable of recomputing the hash, deleting the record

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 167-176
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9999

outright, replaying an old but internally valid state, or tampering with the log that would otherwise reveal any of this. Section 3.6 discusses the implications of each undetected category in turn. Separately, SKN-13 showed that two near-simultaneous authorized updates can silently overwrite one another (a lost update), with the final stored hash remaining fully consistent with the final data and therefore not surfacing as an integrity alarm; this is a data-consistency limitation distinct from the manipulation-detection scenarios above and is likewise carried into Section 3.6. The initial average of under 0.01 seconds per record, from the original five-record pilot, was based on a single measurement per record. To obtain a statistically meaningful estimate, the validation step (database fetch, hash recomputation, and comparison, matching the definition of "validation processing time" used throughout this article) was subsequently timed over 500 repeated runs on one record, following a 30-run warm-up that was discarded, on the same machine described in Section 2.2(c) (Windows 10, Intel Core i3, 8 GB DDR4 RAM). This yielded a mean of 0.4414 ms, a median of 0.4026 ms, a standard deviation of 0.1539 ms, a range of 0.3369 to 2.5871 ms, and a 95% confidence interval of 0.4279 to 0.4549 ms for the mean; the maximum value, well above the mean and median, is consistent with an occasional slower run (e.g., due to background I/O or PHP/MySQL connection overhead) rather than a systematic slowdown, since the median and the 95% CI remain tight and well below 1 ms. This confirms, with statistical support rather than a single reading, that the hash-comparison mechanism imposes negligible computational overhead per record.

3.5 Discussion

The test results show that combining the audit-trail module with SHA-256 hashing effectively ensures data integrity and detects manipulation performed outside the application's official procedure. This effectiveness is primarily supported by the avalanche-effect property of SHA-256 demonstrated in Section 3.2, whereby a two-character change in the address block number causes the entire 256-bit hash value to change completely, so that a mismatch is easily detected through a simple comparison rather than a manual inspection of the data content.

Compared with the blockchain-based approach commonly used in similar studies to secure forensic data [8], [9], [10], [12], [7], the centralized audit-trail-with-hashing approach used in this study offers a substantially simpler implementation, and its validation processing time of only a few milliseconds on standard hardware is consistent with the expectation that avoiding a distributed consensus mechanism reduces computational overhead. This study did not run a controlled head-to-head benchmark against a blockchain-based implementation under equivalent conditions, so "computationally lighter" is presented here as an inferred design advantage grounded in the well-established cost of consensus protocols in the literature, rather than as an experimentally proven comparison; a direct benchmark against a representative blockchain-based baseline is identified as necessary future work to substantiate this claim empirically. The trade-off of this approach is a single point of trust: system integrity depends on the security of the centralized database, unlike blockchain, which distributes trust across many nodes. This limitation is evident in the SKN-03 manipulation scenario, in which the data change was made through direct database access and was only detected once the validation module was executed; the system does not yet prevent an actor with database access from also altering the audit-trail log itself to conceal such traces.

From a practical standpoint, this trade-off is intentional rather than an oversight. A small institution such as a college-level informatics program is unlikely to operate the multiple independent nodes that a blockchain deployment would require, and the operational burden of maintaining such a network would likely exceed the value of the additional tamper-resistance it provides at this scale. The centralized design instead concentrates the engineering effort on making the single point of trust, the database server, easier to secure and monitor, and on providing a credible upgrade path toward hash-chained logging (Section 3.6) rather than a full distributed ledger.

Quantitatively, the result is best read as two separate findings rather than one. The first is a cryptographic property that is essentially guaranteed by SHA-256 itself: given the avalanche effect quantified in Section 3.2 (a mean Hamming distance of 126.42 of 256 bits, 49.38%, SD 8.52, across 60 repeated single-character perturbation trials), any unauthorized change to the protected fields will, with overwhelming probability, produce a detectable mismatch, provided the stored hash itself is not also altered to match. The second is an engineering finding about what the current implementation actually catches in practice: across the expanded 16-scenario test (Table 3), the mechanism reliably detected manipulation that changed data without also updating its stored hash (SKN-02, SKN-03, SKN-07, SKN-09), but did not detect deletion, replay, stored-hash alteration, or audit-log manipulation (SKN-11, SKN-12, SKN-14, SKN-15, SKN-16), for an overall detection rate of 44.4% (4 of 9) among genuine manipulation attempts and zero false positives. This gap between the cryptographic guarantee and the engineering result is the central finding of this study: the avalanche effect guarantees that any data change is detectable in principle, but the present system only checks for it in the narrower case where the attacker leaves the old hash in place, which is why Section 3.6 treats the four undetected categories as concrete, evidence-based priorities for the next development iteration rather than as a hypothetical caveat.

Beyond blockchain, it is also useful to situate this design against other non-blockchain integrity techniques already used in practice, since blockchain is not the only alternative a small institution would realistically consider. Simple checksums (e.g., CRC32) are computationally cheaper than SHA-256 but are designed for accidental-corruption detection, not for resisting a deliberate adversary, since collisions are comparatively easy to construct; they are not an appropriate substitute here. Database-trigger-based auditing, in which the DBMS itself writes change records, can capture changes that bypass the application layer, which is a genuine advantage over the current design, but a trigger-based log stored in the same database is exposed to the same single-point-of-trust weakness identified in this study unless it is also

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 167-176
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9999

replicated or protected externally. Keyed constructions such as HMAC-SHA-256 or digital signatures, and write-once/append-only (WORM) storage or external SIEM-style log shipping, directly address the threat model discussed in Section 2.2(d) by preventing an attacker who can modify application data from silently recomputing a valid protection value; these represent the most directly comparable upgrade path and are the basis for the hash-chaining recommendation in Section 3.6, rather than a move to blockchain.

From a practical-implications standpoint, the prototype is best positioned as a low-cost first step for institutions, such as a small college administrative system, that currently have no cryptographic integrity mechanism at all: adopting audit-trail-with-hashing raises the bar against casual or opportunistic data tampering at a fraction of the engineering and operational cost of a blockchain deployment. It is not yet positioned as a substitute for a full evidentiary chain-of-custody control where legal admissibility depends on demonstrating that the log itself could not have been altered, since that guarantee is precisely what Section 3.6 identifies as still missing. Institutions considering adoption should therefore treat the current system as a detection and monitoring aid rather than as a forensic-grade evidentiary control until the log-protection and authentication gaps are closed.

Several threats to the validity of these findings should be acknowledged. Internally, the small and researcher-constructed test set (five records, two manipulation types) means the reported detection rate could be an artifact of scenario selection rather than a representative sample of real manipulation attempts, and the absence of a formally authenticated user field means the audit trail's attribution accuracy was not itself under test. Externally, all testing was carried out in a single local XAMPP environment, so the timing results in particular may not generalize to a networked, concurrent, or production deployment. Construct-wise, "effectiveness" in this study is operationalized narrowly as hash-mismatch detection under the two tested attack paths; it does not yet cover scenarios in which the attacker also tampers with the audit-trail table itself, which Section 3.6 identifies as the system's principal open threat. These threats do not invalidate the demonstrated mechanism, but they do bound how far the present results can be generalized.

3.6 Research Limitations

a. There is no protection yet against direct changes to the audit-trail table itself. Anyone with database access can technically alter or delete rows in the audit_trail table to conceal traces of manipulation. This was empirically confirmed rather than merely assumed: deleting log entries (SKN-15) and editing a log entry's recorded status (SKN-16) both went completely undetected, and in the latter case the system displayed a contradictory state, an "authorized" badge next to an action still labeled VALIDATION_MISMATCH, without flagging the inconsistency. A hash-chaining mechanism between successive log entries, of the kind formalized for tamper-evident logging by Crosby and Wallach [15], is recommended to close this gap.
b. There is no genuine user-authentication system yet. The user field recorded during testing was selected manually rather than derived from a verified login session.
c. Several categories of manipulation are not yet detected. Expanded testing across sixteen scenarios (Table 3) confirmed three specific gaps beyond log tampering: (i) an attacker who recomputes and overwrites the stored hash together with the data (SKN-14) evades detection entirely, since validation only compares the data against whatever hash is currently stored, whichever party wrote it; (ii) replaying a superseded but internally consistent data-hash pair (SKN-12) is accepted as VERIFIED, because the system checks mathematical consistency between data and hash, not whether that state is still the authorized current one; and (iii) deleting a record outright (SKN-11) leaves nothing for the validation routine to compare, so the manipulation produces no alert at all, and because audit_trail currently cascades on delete from data_identitas, the record's own change history is destroyed along with it. Across all manipulation categories tested, the detection rate was 44.4% (4 of 9 genuine manipulation attempts), with zero false positives on legitimate changes. Closing gaps (i) and (ii) is likely to require binding each hash to a monotonically increasing version or timestamp using a keyed construction (Section 3.5), rather than a bare content hash; closing gap (iii) is likely to require either disallowing hard deletes at the application layer in favor of a logical "revoked" state, or removing the cascading delete so the audit trail survives the record it describes.
d. Concurrent access can silently overwrite legitimate changes. Two near-simultaneous authorized updates to the same record (SKN-13) produced a lost update: the second submission overwrote the first before it was saved, yet the final stored hash remained fully consistent with the final data, so the loss never surfaced as an integrity alarm and was visible only by inspecting the audit trail manually. This is a data-consistency limitation distinct from the manipulation-detection gaps above, and would need row-level locking or optimistic concurrency control (e.g., a version column checked on write) to address.
e. The prototype has no delete feature. Authorized deletion (SKN-10) could not be tested because no delete route exists in the application; the only way a record is currently removed is through direct, unauthorized database access (SKN-11 above). This is a scope gap in the prototype rather than a security flaw, but it means the system's behavior under a legitimate deletion workflow remains unevaluated.
f. The testing environment was local. All tests were conducted in a local development environment (XAMPP) and did not yet cover production conditions such as networked multi-user access at scale or large-scale data loads.

Taken together, these six limitations do not undermine the core empirical finding that SHA-256 hashing reliably detects the specific case of data manipulation that leaves the stored hash unchanged; rather, expanded testing shows that this case

ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9999

is only part of a realistic threat landscape, and that deletion, replay, stored-hash alteration, log tampering, and unsynchronized concurrent writes currently fall outside what the prototype detects. They mark the boundary between what has already been demonstrated at prototype scale, an overall detection rate of 44.4% with zero false positives across the tested manipulation categories, and what would still be required before the system could be relied upon in a full operational forensic setting.

4. CONCLUSION

This study successfully designed and implemented a digital-forensic-based data-change tracking system, the Digital Evidence Integrity System, consisting of three integrated modules, namely audit trail, SHA-256 hashing, and integrity validation, thereby addressing the first research problem and objective. The SHA-256 hashing algorithm was verified to produce a consistent 64-character hexadecimal hash value for identical data content and was shown to be sensitive to even the smallest change through the avalanche effect (mean Hamming distance of 126.42 of 256 bits across 60 repeated trials), addressing the second research objective. Testing began with a five-record pilot, in which the system detected 100% (2 of 2) of the unauthorized update scenarios tested with no false positives, meeting the proposal's achievement indicator of at least 95% for that pilot sample; testing was then expanded to sixteen scenarios across seven manipulation categories, addressing the third research problem and objective more thoroughly and yielding a more representative result: a 44.4% detection rate (4 of 9 genuine manipulation attempts) with zero false positives, the mechanism reliably catching unauthorized changes that leave the stored hash untouched, but not yet catching deletion, replay of a superseded state, an attacker who recomputes the hash itself, or tampering with the audit-trail log. This expanded result, rather than the pilot figure alone, is the one this study treats as representative of the prototype's current detection capability. The centralized audit-trail-with-hashing approach appears computationally lighter than the blockchain approach commonly used in similar studies, based on validation processing times on the order of milliseconds and the absence of a distributed consensus mechanism, an inferred design advantage that would need to be confirmed by a direct benchmark (Section 3.5) before being generalized; the approach nonetheless appears suitable in principle for small- to medium-scale information systems for the narrower class of manipulation it currently detects. At the same time, this study identifies specific, empirically confirmed limitations of the approach: an attacker able to recompute and overwrite the stored hash together with the data evades detection entirely, a superseded but internally consistent data-hash pair can be replayed undetected, an outright record deletion leaves nothing to validate against, and the audit-trail log itself can be deleted or edited without any built-in safeguard, meaning that system integrity still depends on a single point of trust, the centralized database. Future work is therefore directed toward developing a hash-chaining mechanism between audit-trail entries, so that each log entry includes the hash of the previous entry, moving the system toward the concept of a digital chain of custody; binding each stored hash to a monotonically increasing version or timestamp using a keyed construction such as HMAC-SHA-256 to close the replay and stored-hash-alteration gaps; disallowing hard deletes at the application layer (or removing the cascading delete on the audit table) to preserve a record's history even after removal; adding row-level locking or optimistic concurrency control to prevent silent lost updates; and adding a genuine user-authentication mechanism in place of the manually selected user field used during testing.

REFERENCES

[1] E. Casey and E. Casey, “The chequered past and risky future of digital forensics The chequered past and risky future of digital forensics,” *Aust. J. Forensic Sci.*, vol. 00, no. 00, pp. 1–16, 2019, doi: 10.1080/00450618.2019.1554090.
[2] C. Hargreaves, F. Breitinger, L. Dowthwaite, H. Webb, and M. Scanlon, “Forensic Science International : Digital Investigation DFPulse : The 2024 digital forensic practitioner survey,” *Forensic Sci. Int. Digit. Investig.*, vol. 51, no. November, p. 301844, 2024, doi: 10.1016/j.fsidi.2024.301844.
[3] C. Gilbert and M. A. Gilbert, “Exploring Secure Hashing Algorithms for Data Integrity Verification,” vol. 7, no. 11, pp. 373–390, 2025.
[4] Q. Kester and I. B. Senkyire, “Validating of Digital Forensic Images Using SHA-256,” pp. 1–5, 2019.
[5] D. Singh, H. Kaur, C. Verma, N. Kumar, and Z. Illés, “Original article A novel 3-D image encryption algorithm based on SHA-256 and chaos theory,” *Alexandria Eng. J.*, vol. 122, no. March, pp. 564–577, 2025, doi: 10.1016/j.aej.2025.03.026.
[6] F. Frieyadie, “Penggunaan Metode Profile Matching Untuk Sistem Penunjang Keputusan Kenaikan Jabatan Pada Instansi Pemerintah,” *Paradig. - J. Komput. dan Inform.*, vol. 18, no. 2, pp. 75–80, 2016, [Online]. Available: http://ejournal.bsi.ac.id/ejurnal/index.php/paradigma/article/view/1228
[7] H. M. Elgohary and S. M. Darwish, “Improving Uncertainty in Chain of Custody for Image Forensics Investigation Applications,” *IEEE Access*, vol. 10, no. 1, pp. 14669–14679, 2022, doi: 10.1109/ACCESS.2022.3147809.
[8] H. F. Atlam, N. Ekuri, M. A. Azad, and H. S. Lallie, “Blockchain Forensics : A Systematic Literature Review of Techniques , Applications , Challenges , and Future Directions,” *Electronics*, 2024, doi: https://doi.org/10.3390/electronics13173568.
[9] S. Johri, “Strengthening Digital Forensics with Blockchain Technology and Algorithms,” *World J. Adv. Res. Rev.*, vol. 24, no. October, pp. 459–467, 2024.
[10] D. R. Rani, T. Karthik, T. Narasimha, and K. Rajesh, “Blockchain Based Framework for Securing Digital Evidence,” vol. 2, pp. 488–493, 2025, doi: 10.5220/0013885300004919.
[11] O. S. Igonor and M. B. Amin, “The Application of Blockchain Technology in the Field of Digital Forensics : A Literature Review,” 2025.

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 167-176
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9999

[12] D. Batista *et al.*, "Exploring Blockchain Technology for Chain of Custody Control in Physical Evidence : A Systematic Literature Review," 2023.
[13] F. F. Alruwaili, "CustodyBlock : A Distributed Chain of Custody Evidence Framework," 2021.
[14] K. E. N. Peffers, T. Tuunanen, and M. A. Rothenberger, "A Design Science Research Methodology for Information Systems Research," vol. 24, no. 3, pp. 45–77, 2007, doi: 10.2753/MIS0742-1222240302.
[15] S. A. Crosby and D. S. Wallach, "Efficient Data Structures for Tamper-Evident Logging," *Dep. Comput. Sci. Rice Univ.*, 2009.