engineering proceedings

MDPI

Proceeding Paper

Blockchain-Based Framework for Immutable Security Logs: Enhancing Integrity and Traceability †

Daniel Nuñez-Agurto *, Raúl Faz, Eduardo Benavides-Astudillo, John Cruz-Garzón, Javier Cevallos-Farias and Franklin Pérez

Department of Computer Science, Universidad de las Fuerzas Armadas ESPE, Sangolquí 171103, Ecuador; refaz@espe.edu.ec (R.F.); debenavides@espe.edu.ec (E.B.-A.); jjcruz9@espe.edu.ec (J.C.-G.); jjcevallos14@espe.edu.ec (J.C.-F.); frperez4@espe.edu.ec (F.P.)

* Correspondence: adnunez1@espe.edu.ec; Tel.: +593-68416705

† Presented at the Sixth International Conference on Information Systems and Software Technologies (ICI2ST 2026), Quito, Ecuador, 23–25 September 2026.

Abstract

The management and auditing of security logs present a critical challenge in cybersecurity due to the inherent vulnerabilities of traditional systems in terms of integrity, immutability, and traceability. This research develops and implements an innovative solution based on blockchain technology, using Hyperledger Fabric as a permissioned platform to ensure the secure and immutable management of logs. The multi-organization architecture segregates responsibilities between the *LogProvider* (filtering and collection via *rsyslog*) and the *LogAuditor* (cryptographic verification). The solution employs *smart contracts* to define business rules and ensures the integrity of each record through SHA-256 digital signatures. Experimental validation confirmed that the append-only ledger architecture, combined with cryptographic block chaining via SHA-256, provides structural immutability; attempts to modify existing records are rejected at the chaincode validation layer. The implementation achieved an average end-to-end latency of 390 ms under sequential load, sustaining up to 3.16 TPS under concurrent stress with 100% transaction success rate across 455 validated operations. By consolidating segregation of duties and stable performance, the system provides a distributed, tamper-resistant environment for reliable and auditable log management in critical infrastructures.

Keywords: Hyperledger Fabric; security logs; immutability; auditing; blockchain

1. Introduction

The integrity and security of information systems are constantly compromised by sophisticated threats, which demand robust defense mechanisms and infallible traceability [1,2]. In this context, the collection, storage, and immutable verification of critical events and security logs are not only an operational necessity but also an imperative requirement for forensic auditing and regulatory compliance [3,4]. Although security logs play a crucial role, their traditional management presents significant disadvantages, such as vulnerability to tampering, forgery, and record loss, particularly when stored on centralized servers or conventional databases.

Blockchain technology emerges as a new alternative to address these challenges, enabling a functional and decentralized infrastructure, one that is transparent and resistant to manipulation [5–7]. Blockchain, which initially appeared with Bitcoin [8,9], has evolved

Check for updates

Academic Editor: Thomás Borja

Published: 18 September 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

to encompass areas such as cybersecurity, thanks to properties such as immutability, distributed trust, and traceability, supported by cryptographic functions such as hashes and data structures like Merkle Trees [10–13]. Specifically, the incorporation of blockchain to manage security logs not only enhances the integrity of the data stored but also facilitates the auditing of recorded events [14–16]. This allows for cryptographic verification without the need for third parties. Furthermore, the use of consensus algorithms and distributed validation mechanisms ensures that once records are added to the chain, they can no longer be altered.

The integration of blockchain technology to ensure the integrity and traceability of critical records has been an area of intensive research in recent years, with applications extending beyond the financial domain. One significant line of research has focused on irreversible registration and traceability in supply chains. For example, Wang et al. [17] proposed a solution that employs blockchain and smart contracts to immutably record each event and product movement, demonstrating the feasibility of complete transparency in audits. Similarly, Sheng et al. [18] confirmed the authenticity and integrity of product metadata using the Ethereum network, validating blockchain as a viable and secure substitute for paper-based tracking.

A line of research closer to cybersecurity focuses on the direct protection of event logs and digital evidence. Morillo et al. [19] propose recording log event hashes on the blockchain to ensure immutability, chronological traceability, and non-repudiation, recognizing the importance of logs in forensic analysis and regulatory compliance. Likewise, Pourvahab et al. [20] developed a forensic architecture applying blockchain to software-defined networking (SDN) environments to protect logs as evidence, ensuring the Chain of Custody (CoC) through hashing and homomorphic signatures, thereby reinforcing authenticity from collection to analysis.

Additionally, the literature highlights the need for complementary mechanisms of security and access management. Patel et al. [21] demonstrated the value of data preprocessing, using Artificial Intelligence and fuzzy logic to classify information and ensure that only valid and reliable data are recorded on the chain. This preliminary filtering is complemented by the use of smart contracts and digital signatures—tools that, according to the evidence, are essential for managing granular access to sensitive data, ensuring that only authorized users can verify immutable information.

Despite these significant advances validating the potential of blockchain for the integrity, authenticity, and traceability of logs, the current literature still presents a critical gap in the implementation of solutions tailored to high-performance institutional environments. Existing works are often limited to general proof-of-concept studies or public blockchain architectures that are not suitable for the high volume and efficiency requirements demanded by security logs in organizations. In this sense, designing a system that combines blockchain principles with the specific needs of security logs represents an important opportunity to greatly enhance the security of processes, as well as transparency and trust in monitoring and digital auditing procedures.

Specifically, existing proposals report either no throughput benchmarks [19,20] or rely on public Ethereum chains with latencies exceeding 2 s unsuitable for real-time log auditing [6]. In contrast, the present work demonstrates 3.16 TPS sustained throughput with 390 ms mean latency on a permissioned Fabric 2.5 network, validated under concurrent load with dual-organization endorsement.

In this context, the present work was developed, with the main objective of designing and implementing a functional system based on blockchain technology to guarantee the integrity, traceability, and authenticity of security logs. We address this need using the Hyperledger Fabric framework and propose a multi-layer consortium network architecture composed of two key organizations: LogProvider and LogAuditor. The main contributions of this research are as follows:

- A comparative analysis is provided between traditional methods of log storage and the blockchain-based solution, evidencing its improvements in record security.
- An automatic mechanism is designed and implemented for the collection and transmission of logs to the blockchain network, ensuring efficient data transfer from source systems.
- Smart contracts are developed to guarantee the immutable registration of logs, and a web interface is implemented to facilitate querying, validation, and visualization of records, ensuring non-repudiation through cryptographic auditing.

The remainder of this article is structured as follows: Section 2, Materials and Methods, details the requirements analysis, the justification for choosing Hyperledger Fabric, and the proposed consortium architecture for log management. In Section 3, Results and Discussion, the implementation of smart contracts and the validation interface are presented, along with the system's performance evaluation to demonstrate the immutability, traceability, and authenticity of the records. Finally, Section 4, Conclusions, summarizes the main findings of the research and proposes future lines of work.

2. Materials and Methods

The introduction of complex IT systems requires a structured management approach for all steps in the project life cycle, from conception to deployment and continuous improvement. In the present work, the PPDIOO methodology (Prepare, Plan, Design, Develop, Implement, Operate, Optimize) was used, adjusted to the specific characteristics of a blockchain-based solution for managing and verifying security logs [22,23]. This chapter explains, for each step, how the practical application of this methodology was carried out and how it was used to ensure an orderly, effective, and secure evolution in the development of the system.

2.1. Phase 1: Preparation

The initial Preparation phase was dedicated to the critical assessment of traditional methods used for the storage and management of security logs in computer systems. This systematic analysis identified inherent structural deficiencies that compromise the reliability of records, especially in scenarios where irrefutable integrity, traceability, and authenticity are required for auditing and forensic analysis. Methods such as storage in flat files, relational databases, traditional syslog, or the use of local disks are intrinsically susceptible to tampering and falsification of events without leaving detectable evidence.

Table 1 summarizes the findings of this evaluation, highlighting the specific limitations of each method in relation to the principles of immutability and non-repudiation. It is evident that none of these approaches offers strong guarantees against the alteration of security records, which justifies the imperative need to design and implement a robust alternative based on blockchain technology.

https://doi.org/10.3390/engproc2026153014

Table 1. Analysis of traditional methods for security log management.

Ref.	Traditional Method	Description	Integrity Limitations	Traceability Limitations	Authenticity Limitations
[24–26]	Plain text files (flat files)	Records saved in local or server *.log* or *.txt* files.	Vulnerable to tampering without leaving a trace.	Do not allow for reliable verification of the event sequence.	No validation of log origin.
[24,25]	Relational databases (SQL)	Use of DBMS such as MySQL or PostgreSQL to store records.	Records can be edited without a clear history.	Require advanced configurations to ensure traceability.	Lack native digital signature mechanisms.
[27]	Traditional Syslog (RFC 3164)	Standard logging protocol in UNIX systems.	Does not include integrity verification mechanisms.	Does not guarantee exact ordering or secure origin.	Messages can be forged or intercepted.
[28]	Alert email	Sending critical logs to predefined email addresses.	Content can be altered in transit if not encrypted.	No reliable or synchronized timeline.	Difficult to verify sender without digital signatures.
[25,26,28]	Local disk storage	Logs stored directly on local hard drives.	Vulnerable to loss or modification due to failures or access.	Difficult to correlate events without time synchronization.	Lack of verification of event legitimacy.

Once the security and performance requirements were validated, the preparation phase culminated with the critical selection of key technological components and the establishment of a reproducible development environment. These technical decisions laid a foundation consistent with the system's objectives of immutability, traceability, and high performance.

- Consortium Platform: Hyperledger Fabric [29,30] was selected as the permissioned *blockchain* platform, using the native tools *peer*, *configtxgen*, and *cryptogen* compiled with the Go programming language. Hyperledger Fabric was selected based on four technical criteria:
  - Permissioned access via MSP ensures only LogProvider and LogAuditor can submit/validate transactions.
  - Native Raft consensus provides deterministic block ordering with 2 s batch timeout.
  - Chaincode supports custom validation logic impossible in append-only databases.
  - X.509 certificate infrastructure aligns with NIST SP 800-57 key.
- Orchestration and Environment: *Docker* and *Docker Compose* were employed for the orchestration of distributed services [31]. The base environment was organized via the Windows Subsystem for Linux (WSL2) with Ubuntu 22.04.5 LTS, providing an optimized native Linux environment.
- Smart Contract Development: JavaScript was chosen for the *smart contracts* (*chaincode*), leveraging the *Node.js runtime* and *npm*, which facilitates client integration and development.
- Data Integration (*Syslog*): For real-time data capture, integration with *rsyslog* was established using the UDP protocol on port 5140 [32,33]. This enables direct collection of operating system logs and implements rule-based filtering of security events, targeting the same four event categories defined in the collection module configuration.

- Security and Protocols: The security infrastructure was based on *OpenSSL* for cryptography and *TLS* certificate management, ensuring the confidentiality of internal communications. Protocol support includes HTTP/HTTPS for interfaces, *gRPC* for Fabric communication, and UDP for initial log capture.

The selected technology stack establishes a reproducible development environment that facilitates both iterative development and rigorous validation in subsequent phases.

2.2. Phase 2: Planning

The planning phase focused on defining the solution architecture and the conceptual design of the data flow, ensuring alignment between the selected technological platform (Hyperledger Fabric) and the requirements of integrity and non-repudiation of the records.

Initially, a detailed analysis of Fabric's internal architecture was carried out, including its nodal components (peers, orderers, Certificate Authorities), the transaction life cycle, and the logic of smart contracts (chaincode). This preliminary study was crucial to map Fabric's native functionalities to the requirements of log management:

- Network Governance: A consortium network topology was planned to ensure access control and privacy of enterprise logs, limiting participants to the designated organizations (LogProvider and LogAuditor).
- Chaincode Design: The essential functions of the smart contract (registration, query, and validation) were defined, and the minimum data format that the chaincode would store on the ledger was established (primarily the log hash, essential metadata, and timestamp).
- Data Integration: The use of the Syslog protocol was planned for real-time log collection, prioritizing the UDP architecture and defining log hashing at the collection point to ensure that the event's fingerprint is immutable before being sent to the blockchain.

Finally, the project roadmap was formalized by structuring the timeline according to the phases of the PPDIOO methodology, as detailed in Table 2.

Table 2. Planning for blockchain-based log assurance system.

Phase	Month 1	Month 2	Month 3	Month 4
Preparation	X			
Planning	X			
Network design		X		
Implementation		X	X	
Operation			X	
Optimization				X

Note: 'X' indicates the active execution period for each phase.

2.3. Phase 3: Design

The design phase was devoted to defining a permissioned network architecture based on *Hyperledger Fabric* that would guarantee the integrity, traceability, and authenticity of security *logs*. This architecture was configured to operate as a Proof of Concept (PoC) with scalability to corporate environments. Figure 1 shows the minimalist network topology.

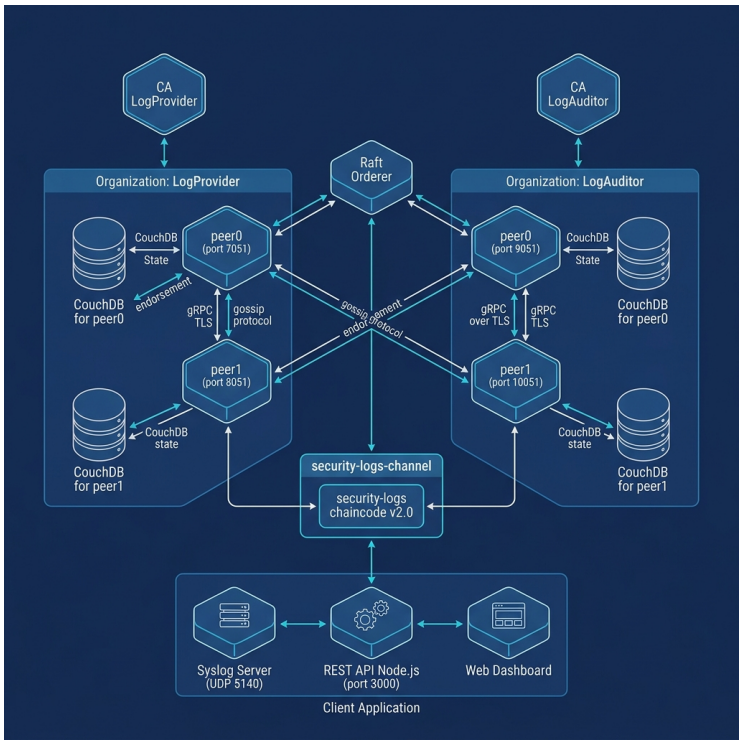


Figure 1. Fabric network architecture for security logs. The arrows represent communication and transaction data flows.

2.3.1. Consortium Architecture and Segregation of Duties

The consortium is composed of four actors with distinct roles: the *LogProvider* generates and signs security logs; the *LogAuditor* independently validates records without the capability to modify them; the *Orderer* runs Raft consensus and creates blocks; and the *Chaincode* enforces validation, storage, and query logic. The separation of LogProvider and LogAuditor prevents any single entity from both generating and approving log records.

Each organization was provisioned with *two Peers* (*peer0* and *peer1*), a configuration that satisfies essential redundancy criteria, ensuring availability against node failures and performance, meeting the minimum for distributed validation. For record management, a dedicated channel, *security-logs-channel*, was configured, providing isolation and confidentiality for handling sensitive data.

The ordering service employs Raft (`etcdraft`) consensus, a leader–follower protocol tolerant to f node failures with $2f + 1$ total nodes. The orderer is configured with `BatchTimeout=2s` and `MaxMessageCount=10`, meaning blocks are cut every 2 s or when 10 transactions accumulate—whichever occurs first. This configuration directly determines the system throughput ceiling of ~3.5 TPS observed experimentally.

2.3.2. Processing Pipeline and Smart Contract Design

The system is complemented by a robust client application that establishes a complete *pipeline* for processing *logs* prior to registration on the blockchain.

Module 1: Collection and Filtering (*syslog-collector.js*)

Acts as a specialized Syslog server that collects *logs* via the *UDP 5140* protocol. This component applies rule-based filtering defined in /etc/rsyslog.d/99-blockchain.conf, targeting four event categories: severity WARNING or lower, SSH authentication events (auth/authpriv facility), sudo and PAM commands, and critical kernel errors.

Module 2: Processing and Linking (*app.js*)

Implemented as a *REST API* on port 3000 in Node.js. It is responsible for: (1) structuring the *log* in JSON format; (2) applying *Base64* encoding; (3) generating the cryptographic *SHA-256* signature of the *log* and preparing it for submission; and (4) invoking the *chaincode* using *gRPC over TLS*.

Module 3: Audit Interface (*dashboard.html*)

Provides real-time visualization of stored *logs*, including critical auditing functionalities such as automatic Base64 decoding, advanced filters, and, primarily, the capability to verify data integrity through cryptographic control mechanisms.

Smart Contract (*security-logs v2.1*)

Deployed on the channel, it contains the business logic to manage the log life cycle. Each record stored on the ledger includes the complete log payload encoded in Base64, comprising logId, timestamp, logType, severity, userId, sourceIP, action, resource, and message, alongside a SHA-256 hash that serves as an integrity fingerprint for tamper detection. This design prioritizes forensic completeness and auditability over storage efficiency, with each record consuming approximately 1–2 KB of ledger space.

2.4. Phase 4: Implementation

The network was deployed via Docker Compose orchestrating 11 services, as detailed in Table 3. Cryptographic artifacts were generated using `cryptogen` and `configtxgen`. Smart contract `security-logs v2.1` (JavaScript) was deployed with dual-org endorsement policy requiring signatures from both LogProviderMSP and LogAuditorMSP per transaction.

Table 3. Docker services and configuration for the LogNetwork deployment.

Component	Technology	Configuration	Purpose
Orderer	Fabric orderer (etcdraft)	Port 7050; BatchTimeout = 2 s; MaxMessageCount = 10	Block creation and consensus
CA LogProvider	Fabric CA	Port 7054	Issue certificates for LogProvider MSP
CA LogAuditor	Fabric CA	Port 8054	Issue certificates for LogAuditor MSP
peer0.logprovider	Fabric peer	Port 7051; CouchDB 5984	Endorse/commit transactions
peer1.logprovider	Fabric peer	Port 8051; CouchDB 6984	Redundancy for LogProvider MSP
peer0.logauditor	Fabric peer	Port 9051; CouchDB 7984	Independent validation
peer1.logauditor	Fabric peer	Port 10051; CouchDB 8984	Redundancy for LogAuditor MSP
couchdb0.logprovider	CouchDB	Port 5984	World State for peer0.logprovider
couchdb1.logprovider	CouchDB	Port 6984	World State for peer1.logprovider
couchdb0.logauditor	CouchDB	Port 7984	World State for peer0.logauditor
couchdb1.logauditor	CouchDB	Port 8984	World State for peer1.logauditor

2.5. Phase 5: Operation and Experimental Evaluation

The operation phase focused on deploying the system in a functional environment and executing a structured experimental evaluation protocol. Network stability was verified through peer diagnostic queries confirming blockchain synchronization, channel availability,

and correct application of endorsement policies. Functional validation confirmed real-time log processing and cryptographic integrity: only logs whose SHA-256 fingerprint matched the immutably recorded hash were approved by the system. Performance was quantified through controlled load tests measuring end-to-end storage latency from REST API receipt to ledger confirmation, and query-time latency via CouchDB indexes, with per-request timing, success rate, and queue depth recorded across all concurrency levels.

It is important to note that all experiments were conducted in a single-machine Docker environment, where inter-node communication occurs over a local bridge network with negligible latency. Consequently, the reported latency values represent a lower bound that excludes the network overhead inherent in geographically distributed deployments. Additionally, storing full log payloads on-chain, rather than only hashes, accelerates ledger growth; long-term storage costs in production environments should be evaluated against organizational retention policies.

3. Results and Discussion

This section presents and analyzes the empirical results obtained from the experimental evaluation of the implemented blockchain solution. The results demonstrate the successful fulfillment of the system's specific objectives, quantifying transactional efficiency, validating cryptographic integrity, and assessing resilience under concurrent load. The analysis contextualizes these findings in relation to the immutability and auditing requirements of security logs.

3.1. Hash Algorithm Benchmark

SHA-256 was selected based on the benchmark results and four key criteria. First, it demonstrated the lowest mean latency across all tested payload sizes, ranging from 8.77 µs at 256 bytes to 14.64 µs at 4096 bytes, with stable throughput exceeding 68,000 ops/s in all cases. Second, its broad hardware acceleration support ensures deterministic performance under concurrent load. Third, it presents a mature compliance posture suitable for government deployments and auditability requirements. Fourth, and critically, since SHA-256 is the native hashing algorithm of Hyperledger Fabric for internal block hashing, transaction Merkle tree construction, and asset identifier generation, its use in the chaincode eliminates format incompatibilities and conversion overhead between hash schemes. Together, these results justify SHA-256 as the optimal choice for the security log integrity pipeline, as shown in Table 4.

Table 4. Hash benchmark across candidate algorithms.

Input (Bytes)	SHA-256 (µs)	SHA-3-256 (µs)	BLAKE2b (µs)	SHA-256 Throughput (ops/s)
256	8.77	178.63	46.64	113,981
1024	8.84	190.17	20.19	113,107
4096	14.64	775.36	12.18	68,329

3.2. Throughput and Latency Under Concurrent Load

Latency variability increased with concurrency, with the estimated standard deviation rising from 184 ms to 2349 ms as queueing pressure increased, as shown in Table 5.

https://doi.org/10.3390/engproc2026153014

Table 5. Throughput and latency under varying concurrency.

Conc.	Req.	TPS	Succ.	Mean	Std. Dev. *	Min	P50	P95	Max
1	70	2.33	100%	390 ms	184 ms	224 ms	355 ms	657 ms	841 ms
2	92	3.05	100%	595 ms	197 ms	330 ms	566 ms	890 ms	1024 ms
5	97	3.16	100%	1434 ms	420 ms	672 ms	1441 ms	2132 ms	2446 ms
10	102	3.00	100%	2883 ms	988 ms	892 ms	2891 ms	4516 ms	5620 ms
20	94	2.94	100%	6295 ms	2349 ms	1735 ms	6369 ms	10,233 ms	11,878 ms

* Estimated standard deviation derived from the P95–P50 interval, assuming an approximately normal latency distribution ($\sigma \approx$ (P95–P50)/1.645).

Transactional performance was evaluated using concurrent HTTP requests to the log registration endpoint (`POST/api/logs`). Each concurrency level (1, 2, 5, 10, and 20 clients) ran for 30 s after a 10-request warm-up, measuring end-to-end latency from submission to ledger confirmation. A total of 455 transactions were validated.

Throughput remained near 3.16 TPS across all concurrency levels, limited by the Raft configuration (`BatchTimeout=2s`, `MaxMessageCount=10`), which generates a block every two seconds or after ten transactions. Increasing concurrency beyond the block capacity increased queueing rather than throughput. Mean latency rose from 390 ms to 6295 ms, while the minimum latency of 224 ms reflects the baseline time required for endorsement, ordering, and ledger commit. Transaction success remained at 100%, indicating stable operation under load.

Raft parameters represent a configurable trade-off between throughput and latency. In this study, the selected configuration provides an appropriate balance for secure log management, where audit integrity takes precedence over real-time performance. In scenarios with higher log ingestion rates, these parameters can be adjusted to increase throughput at the cost of higher per-transaction latency.

3.3. Query Scalability

To evaluate system behavior under ledger growth, the response time of the `GET/api/logs` endpoint was measured, which returns the complete set of stored records. Measurements were performed across three ledger sizes accumulated progressively through successive rounds of the load test, executing 10 consecutive queries per size and reporting results in milliseconds, as summarized in Table 6.

Table 6. Query latency as ledger size increases.

Ledger Size (Docs)	Mean Latency	Min	Max	vs. Baseline
449	459 ms	385 ms	706 ms	—
972	620 ms	469 ms	693 ms	+35%
1437	705 ms	535 ms	915 ms	+54%

The results show that query latency grows sub-linearly with respect to the volume of stored documents. With 449 records, a mean latency of 459 ms was obtained, increasing to 620 ms with 972 records and to 705 ms with 1437 records. This behavior implies that, although the ledger grew by 220% between the initial and final easurements, latency increased by only 54% relative to the baseline, evidencing that CouchDB optimizes full scans through internal indexing mechanisms. The latency range observed in each scenario, characterized by stable minimum and maximum values, confirms the consistency of the system under repeated queries over the same ledger state.

3.4. Fault Tolerance

To evaluate system resilience under node failure conditions, real faults were simulated by stopping Docker containers while the system actively processed transactions. Four scenarios were assessed: normal operation with all four peers active, failure of a non-anchor peer in the LogProvider organization, simultaneous failure of a non-anchor peer in both organizations, and automatic recovery following container restart. The outcomes of these scenarios are summarized in Table 7.

Table 7. Fault tolerance scenarios and outcomes.

Scenario	Result	Latency	Notes
Normal (4 peers)	✓	~400 ms	Baseline
peer1.logprovider down	✓	467 ms	+16% overhead
peer1 both orgs down	✓	418 ms	2/4 peers activos
Recovery post-restart	✓Auto	—	Gossip sync at block #294

Note: The checkmark symbol (✓) indicates successful transaction validation and block commitment under the specified fault scenario.

The results confirm that the system maintained full operational capacity across all failure scenarios. The loss of `peer1.logprovider` introduced a latency overhead of only 16% (467 ms versus 400 ms baseline), as endorsement continued through the remaining active peers. The simultaneous failure of non-anchor peers in both organizations yielded a latency of 418 ms, remaining within operational bounds with 2 of 4 peers active. Upon container restart, recovered peers automatically resynchronized with the network via the Gossip protocol, reaching canonical ledger state at block #294 without manual intervention or data loss. Unlike traditional replicated databases where primary node failure requires manual failover, Hyperledger Fabric sustains operation under any combination of non-anchor peer failures with automatic state recovery, a property that is absent in centralized database architectures.

3.5. Functional Validation

The functional validation of the system confirmed the correct implementation of the consortium architecture and the blockchain network. The security-logs-channel was verified as operational, with participating peers successfully joined to the channel (Listing 1). Integrity validation confirmed that the system approves only logs whose SHA-256 fingerprint matches the hash immutably recorded on the blockchain; even the alteration of a single character resulted in immediate rejection. The Syslog collection module applied rule-based filtering via rsyslog configuration, ensuring that only critical security events—SSH authentication failures, sudo executions, and kernel errors—were processed and transmitted to the blockchain.

Listing 1. Peer participation in the security channel.

```
$ docker exec -it cli bash
$ cd/opt/gopath/src/github.com/hyperledger/fabric/peer
$ peer channel list

Channels peers has joined:
security-logs-channel
```

The immutability of records was demonstrated through the functional distinction between the World State and the blockchain. A security log was generated, stored, and its presence in the World State verified through a query. Subsequently, the DeleteSecurityLog

chaincode function removed the record from the World State. The subsequent verification, shown in Listing 2, confirms both the successful deletion of the key from the World State and the persistence of the original transaction in the blockchain through the GetBlockByTxID function of the qscc system chaincode. These results demonstrate that although the mutable state can be updated, the blockchain preserves an immutable, cryptographically verifiable transaction history.

Listing 2. Verification of blockchain immutability after record deletion.

```
$ peer chaincode invoke -o orderer1.log-network.com:7050 \
-C security-logs-channel -n security-logs \
-c '{"function":"DeleteSecurityLog",
"Args":["REAL_SYSLOG_1753629420230_92"]}'

Chaincode invoke successful.
Log REAL_SYSLOG_1753629420230_92 deleted successfully

$ peer chaincode query -C security-logs-channel -n qscc \
-c '{"Args":["GetBlockByTxID",
"security-logs-channel",
"REAL_SYSLOG_1753629420230_92"]}'

Transaction ID : REAL_SYSLOG_1753629420230_92
Block Number   : 8
Validation     : VALID
Timestamp      : 2025-06-27T15:14:07Z
```

3.6. Threat Model and Security Analysis

The security analysis is grounded in a formal threat model that defines the assumed adversary and the structural defense mechanisms of the system. The adversary is assumed to have network access to the LogProvider boundary, the ability to submit malformed logs or replay prior valid transactions, and limited insider access to a single organization without inter-organizational collusion, as shown in Table 8.

Table 8. Attack vectors and mitigations.

Attack Vector	Detail	Mitigation/Note
Post-submission tampering	Alteration of already confirmed records on the ledger	Hash chaining and chaincode validation reject modified payloads
Transaction replay	Resubmission of prior valid logs	Unique identifiers and timestamp verification prevent duplicate acceptance
Single-organization endorsement abuse	One organization approves its own transactions	Dual-organization endorsement policy requires both MSP signatures
Peer compromise	Compromised peer with inconsistent ledger state	Gossip protocol automatically restores canonical ledger state
Temporal synchronization	Strict log ordering assumes synchronized clocks across nodes	Known limitation outside system scope

Table 8. *Cont.*

Attack Vector	Detail	Mitigation/ Note
Payload privacy	Log contents not encrypted at chaincode layer	Known limitation—payload encryption proposed as future work
Inter-organizational collusion	Simultaneous compromise of both *LogProvider* and *LogAuditor*.	The endorsement policy assumes organizational independence; compromising both organizations would invalidate independent validation. This risk is mitigated by their administrative and physical separation.

4. Conclusions

This study successfully developed an immutable security log system based on Hyperledger Fabric, fulfilling the objectives of ensuring the integrity, traceability, and authenticity of records. This technical solution overcomes the vulnerabilities of manipulation and unauthorized deletion inherent in conventional centralized approaches, providing a replicable model for critical infrastructures, scalable within the throughput constraints imposed by Raft consensus. The validation of architectural immutability was the fundamental achievement; the implementation of a permissioned network with Hyperledger Fabric 2.5 (etcdraft consensus, X.509 certificates, and TLS encryption) proved to be a viable technical solution. Tests confirmed that although records were removed from the World State (CouchDB) for operational purposes, the trace of the original transaction remained unalterable in the blockchain, ensuring the non-repudiation of records.

Experimental evaluation on the LogNetwork testbed confirmed the system's transactional and operational efficiency across five dimensions. End-to-end transaction latency reached 390 ms under sequential load, sustaining up to 3.16 TPS under concurrent stress with 100% transaction success rate across 455 validated operations. The hash algorithm benchmark demonstrated that SHA-256 offers the greatest benefit among the evaluated algorithms, achieving a mean throughput of 93,948 ops/s. Furthermore, since it is the native hashing algorithm of Hyperledger Fabric used internally for block hashing and Merkle tree construction, it eliminates format incompatibilities and conversion overhead Query latency scaled sub-linearly with ledger growth, increasing from 459 ms at 449 documents to 705 ms at 1437 documents, representing only a 54% latency increase against a 220% growth in stored records. Fault tolerance evaluation confirmed that the system remains fully operational under 50% node failure, with automatic ledger resynchronization via the Gossip protocol upon peer recovery. Furthermore, the dual-organization endorsement model structurally eliminates single points of trust at the endorsement layer, a property absent in replicated database solutions, though the single Raft orderer node represents a remaining centralization point that would require a multi-orderer configuration to fully eliminate it.

The system demonstrated advanced auditing and data management capabilities. The log collection mechanism applied rule-based filtering to ensure that only relevant security events consumed resources. The interface exposed through the REST API and Web Dashboard facilitated the querying, validation, and visualization of intact records, consolidating the segregation of duties between LogProvider and LogAuditor and enabling integration with external applications.

As the most relevant future work, the implementation of a geographically distributed multi-node and multi-organization architecture is considered to increase availability, fault tolerance, and enable cross-validation of logs among collaborating entities. Complementarily, integration with existing SIEM platforms such as Splunk or IBM QRadar will be

pursued through the development of standardized connectors, and a multi-chain approach will be explored, where different types of records are stored in separate channels or chains to optimize scalability and data confidentiality.

Author Contributions: Conceptualization, D.N.-A. and E.B.-A.; methodology, D.N.-A., J.C.-F. and R.F.; software, J.C.-G. and F.P.; validation, F.P. and J.C.-G.; formal analysis, J.C.-F. and R.F.; investigation, D.N.-A.; data curation, D.N.-A., J.C.-G. and F.P.; writing—original draft, D.N.-A. and R.F.; writing—review and editing, E.B.-A. and R.F.; supervision, D.N.-A. and R.F.; funding acquisition, D.N.-A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors would like to thank the Distributed Systems, Cybersecurity, and Content Research Group (RACKLY) at Universidad de las Fuerzas Armadas ESPE for their scientific and collaborative contribution.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Nuñez-Agurto, D.; Ron, M.; Carrera, E.V.; Tapia, F.; Cruz, H.; Recalde, L.; Fuertes, W. Design of an Academic CSIRT—A Proposal Based on Strategic Planning Principles. In *Proceedings of the Emerging Research in Intelligent Systems*; Botto-Tobar, M., Cruz, H., Díaz Cadena, A., Durakovic, B., Eds.; Springer: Cham, Switzerland, 2022; pp. 295–308. [CrossRef]
2. López-Pimentel, J.C.; Morales-Rosales, L.A.; Monroy, R. RootLogChain: Registering Log-Events in a Blockchain for Audit Issues from the Creation of the Root. *Sensors* **2021**, *21*, 7669. [CrossRef]
3. Birk, D.; Wegener, C. Technical Issues of Forensic Investigations in Cloud Computing Environments. In *Proceedings of the 2011 Sixth IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering*; IEEE: Piscataway, NJ, USA, 2011; pp. 1–10. [CrossRef]
4. Cho, G.; Go, Y.; Kim, M. A Hyperledger Fabric-Based SBOM Management System for Secure Software Supply Chain Integrity. *Electronics* **2026**, *15*, 2573. [CrossRef]
5. Zhao, W. On Blockchain: Design Principle, Building Blocks, Core Innovations, and Misconceptions. *IEEE Syst. Man Cybern. Mag.* **2022**, *8*, 6–14. [CrossRef]
6. Zhao, W.; Aldyaflah, I.M.; Gangwani, P.; Joshi, S.; Upadhyay, H.; Lagos, L. A Blockchain-Facilitated Secure Sensing Data Processing and Logging System. *IEEE Access* **2023**, *11*, 21712–21728. [CrossRef]
7. Wijesekara, P.A.D.S.N.; Gunawardena, S. A Review of Blockchain Technology in Knowledge-Defined Networking, Its Application, Benefits, and Challenges. *Network* **2023**, *3*, 343–421. [CrossRef]
8. Satoshi, N. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 14 December 2025).
9. Giungato, P.; Rana, R.; Tarabella, A.; Tricase, C. Current Trends in Sustainability of Bitcoins and Related Blockchain Technology. *Sustainability* **2017**, *9*, 2214. [CrossRef]
10. Crosby, M.; Pattanayak, P.; Verma, S.; Kalyanaraman, V. Blockchain Technology: Beyond Bitcoin. *Appl. Innov.* **2016**, *2*, 6–10.
11. Shiho, K. Blockchain for a Trust Network Among Intelligent Vehicles. *Adv. Comput.* **2018**, *111*, 43–68. [CrossRef]
12. Xie, J.; Yu, F.R.; Huang, T.; Xie, R.; Liu, J.; Liu, Y. A Survey on the Scalability of Blockchain Systems. *IEEE Netw.* **2019**, *33*, 166–173. [CrossRef]
13. Adhikari, N.; Ramkumar, M. IoT and Blockchain Integration: Applications, Opportunities, and Challenges. *Network* **2023**, *3*, 115–141. [CrossRef]
14. Zheng, Z.; Xie, S.; Dai, H.; Chen, X.; Wang, H. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In *Proceedings of the 2017 IEEE International Congress on Big Data (BigData Congress)*; IEEE: Piscataway, NJ, USA, 2017; pp. 557–564. [CrossRef]
15. Wang, S.; Ouyang, L.; Yuan, Y.; Ni, X.; Han, X.; Wang, F.Y. Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends. *IEEE Trans. Syst. Man Cybern. Syst.* **2019**, *49*, 2266–2277. [CrossRef]
16. Li, W.; He, M.; Haiquan, S. An Overview of Blockchain Technology: Applications, Challenges and Future Trends. In *Proceedings of the 2021 IEEE 11th International Conference on Electronics Information and Emergency Communication (ICEIEC)*; IEEE: Piscataway, NJ, USA, 2021; pp. 31–39. [CrossRef]

17. Wang, S.; Li, D.; Zhang, Y.; Chen, J. Smart Contract-Based Product Traceability System in the Supply Chain Scenario. *IEEE Access* **2019**, *7*, 115122–115133. [CrossRef]
18. Sheng, S.W.; Wicha, S. Blockchain-Based Traceability for Teak Identity: A Transformational Approach. *J. Mob. Multimed.* **2024**, *20*, 267–306. [CrossRef]
19. Morillo Reina, J.D.; Mateo Sanguino, T.J. Decentralized and Secure Blockchain Solution for Tamper-Proof Logging Events. *Future Internet* **2025**, *17*, 108. [CrossRef]
20. Pourvahab, M.; Ekbatanifard, G. An Efficient Forensics Architecture in Software-Defined Networking-IoT Using Blockchain Technology. *IEEE Access* **2019**, *7*, 99573–99588. [CrossRef]
21. Patel, N.; Patel, D.; Jadav, N.K.; Rathod, T.; Tanwar, S.; Pau, G.; Sharma, G.; Alqahtani, F.; Tolba, A. Fuzzy-Enhanced Secure Messaging Framework for Smart Healthcare System. *IEEE Access* **2024**, *12*, 102977–102993. [CrossRef]
22. Yulianaa, D.; Mogia, I.K.A. Computer Network Design Using PPDIOO Method with Case Study of SMA Negeri 1 Kunir. *J. Elektron. Ilmu Komput. Udayana* **2020**, *2301*, 5373. [CrossRef]
23. Elrashdi, A.S.; Alferjani, S.K.; Omar, R.R.; Hasan, F.M. The efficiency of using PPDIOO Methodology to Design Graduation Projects for Network Department Students. In *Proceedings of the 2024 IEEE 7th International Conference on Advanced Technologies, Signal and Image Processing (ATSIP)*; IEEE: Piscataway, NJ, USA, 2024; Volume 1, pp. 438–442. [CrossRef]
24. Casey, E. *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet*; Academic Press: Cambridge, MA, USA, 2011.
25. Bishop, M. *Computer Security: Art and Science*; Addison-Wesley: Boston, MA, USA, 2003.
26. Marty, R. *Applied Security Visualization*; Addison-Wesley: Boston, MA, USA, 2008.
27. Gerhards, R. The Syslog Protocol. IETF RFC 5424. 2009. Available online: https://tools.ietf.org/html/rfc5424 (accessed on 14 December 2025).
28. Kent, A.; Souppaya, M. *Guide to Computer Security Log Management*; Technical Report Special Publication 800-92; NIST: Gaithersburg, MD, USA, 2006.
29. Li, W.; Guo, H.; Nejad, M.; Shen, C.C. Privacy-Preserving Traffic Management: A Blockchain and Zero-Knowledge Proof Inspired Approach. *IEEE Access* **2020**, *8*, 181733–181743. [CrossRef]
30. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the EuroSys '18: Proceedings of the Thirteenth EuroSys Conference, New York, NY, USA, 23–26 April 2018*; EuroSys '18; Association for Computing Machinery: New York, NY, USA, 2018. [CrossRef]
31. Reis, D.; Piedade, B.; Correia, F.F.; Dias, J.P.; Aguiar, A. Developing Docker and Docker-Compose Specifications: A Developers' Survey. *IEEE Access* **2022**, *10*, 2318–2329. [CrossRef]
32. Zhang, S.; Liu, Y.; Meng, W.; Bu, J.; Yang, S.; Sun, Y.; Pei, D.; Xu, J.; Zhang, Y.; Song, L.; et al. Efficient and Robust Syslog Parsing for Network Devices in Datacenter Networks. *IEEE Access* **2020**, *8*, 30245–30261. [CrossRef]
33. Ghaffarinejad, A.; Syrotiuk, V.R. OpenFlow versus Commercial Load Balancers in a Campus Network. In *Proceedings of the 2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*; IEEE: Piscataway, NJ, USA, 2015; pp. 1–5. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.