A framework for verifying the integrity of IoT logs and tracing tampering based on deterministic Merkle trees

Dongdong Liu[1,2,*], Fuqiang Li[1], Shuo Fang[1]
[1]School of Computer and Information Engineering, Fuyang Normal University, Fuyang 236037, Anhui, China
[2]Anhui Engineering Research Center for Intelligent Computing and Information Innovation, Fuyang Normal University, Fuyang 236037, Anhui, China
Corresponding author: Dongdong Liu (e-mail: fynuldd13965576799@163.com).

ABSTRACT In IoT scenarios such as industrial gateways, park security, vehicle terminals, and medical monitoring, logs are often the first materials retrieved for fault recovery and security evidence, but they are not naturally reliable. Device disconnection, gateway caching, platform script misoperation, and even intruders who obtain maintenance privileges may cause record loss, rewriting, disorder, or field rewriting. This article proposes an IoT log integrity verification and tamper tracing framework based on deterministic Merkle trees to address this issue. The framework first converts heterogeneous logs into recalcitrable canonical objects, and then establishes an evidence chain using leaf hashing, batch root hashing, external anchoring, and audit paths. The article further presents the collaborative process of devices, gateways, platforms, and audit nodes, discussing inclusion verification, consistency verification, anomaly localization, and responsibility boundary division. The research believes that this method does not need to publicly link all logs, which can improve the credibility of logs with lower storage and transmission costs, and provide an executable solution for the safe operation and post investigation of the Internet of Things.

INDEX TERMS Deterministic Merkle tree; IoT logs; Integrity verification; Tampering with traceability

I. INTRODUCTION

A. RESEARCH BACKGROUND

In many IoT projects, logs are initially just records that maintenance personnel casually check when troubleshooting; After the expansion of the system scale, it gradually became the basic material for attack restoration, compliance auditing, and responsibility determination. Industrial control equipment, park cameras, vehicle terminals, medical monitoring devices, and warehouse sensors will continuously generate logs covering status, alarms, instructions, access, and maintenance operations. The NIST guidelines on log management state that organizations need to establish a log infrastructure and continuously complete collection, protection, analysis, and archiving. The problem is that these logs are scattered among devices, gateways, edge servers, and cloud platforms. As long as a layer is deleted, rewritten, delayed in reporting, or batch forged, subsequent investigations will lose stable basis.

The trustworthy boundary of IoT logs is more difficult to distinguish than that of ordinary business systems. Many terminals do not have a stable power supply and do not have enough space to store long-term logs; The equipment is installed in workshops, warehouses, roads, or weak electrical rooms in buildings, and debugging ports, weak passwords, and wireless links may all become tampering entrances. In order to save bandwidth, the gateway also caches, filters, or merges records, resulting in different time points left by the same event on the device side, gateway side, and platform side. Time stamp drift, missing fields, and sequence changes are not uncommon in engineering sites, and these phenomena may arise from normal network jitter or mask human rewriting. The NIST IoT Security Guidelines state that when IoT devices are connected to organizational systems, security requirements need to be clearly defined from the perspectives of device capabilities, manufacturer support, and system risks [2]. This also indicates that log integrity cannot be solely verified on the platform side after the fact.

In practical engineering, log protection is often implemented in several separate ways: some units centralize log writing and only append storage, some rely on database auditing or SIEM alarms, some add digital signatures to critical records, and some projects attempt to write summaries on the chain. These methods each have their own value, but they are not always easy to apply in the context of the Internet of Things. Only adding storage cannot prevent people with backend permissions from changing historical indexes; Signing each item will push the key and computing power pressure onto low-end devices; Uploading the entire log to the blockchain will bring costs and privacy risks. In

contrast, the Merkle tree is more like a compromise tool: it does not move all logs, only uses the root hash and path to prove that a certain record has appeared in a batch, which is suitable for edge gateways to batch process by window.

B. RESEARCH SIGNIFICANCE

From a technical perspective, the Merkle tree is not a structure that only emerged for the Internet of Things. Merkle's early research on hash based signatures has demonstrated the idea of using digest trees to organize authentication information [3]; The Certificate Transparency Protocol has made auditable mechanisms for appendable logs, tree head signatures, inclusion proofs, and consistency proofs [4], [5]. These experiences are of great reference value for IoT logs: the log body can be kept locally, and only batch roots and necessary proofs can be saved externally; Auditors do not need to see all sensitive content to determine whether a record has entered a sealed batch.

The research value of this article can be understood from practical usage scenarios. For data governance, standardized rules can reduce the embarrassment of "the same log has different hashes in different systems"; For security operations, the batch root and audit path move the verification action forward to the device generation, gateway aggregation, and platform storage stages, so exceptions do not have to wait until the accident review to be exposed; For the evidence chain, the system not only indicates that the log has changed, but also further explains which batch, level, and responsibility window the change occurred in roughly.

C. RESEARCH APPROACH

The writing of this article starts from a very specific question: how do auditors determine whether a suspicious log is already in a certain batch when they receive it, and how do they trace the possible problematic links. Regarding this issue, the article first provides the minimum field set and standardization requirements for logs, so that different devices, languages, and platforms can recalculate the same leaf hash; Subsequently, we will discuss how edge gateways can construct deterministic Merkle trees, clarify sorting, odd node handling, and domain separation rules; Redesign to include performance verification, consistency verification, and cross node sampling; Finally, convert the verification failure results into traceability evidence to serve the investigation of the incident and the delineation of responsibility boundaries.

II. THEORETICAL BASIS AND KEY TECHNOLOGIES

A. CONNOTATION OF IOT LOG INTEGRITY

When discussing log integrity, one cannot simply ask if the file has been modified. A more practical issue is: whether the fields of this record are complete, whether the sequence of events can be restored, whether the source device is trustworthy, whether there are any records that have been extracted from the same batch, and whether the evidence chain is closed when referenced in the future. If one item is missing from the content, order, source, batch boundary, and evidence chain, the investigation conclusion will become weaker. For example, if the field of a control instruction remains unchanged, but the device serial numbers before and after it are disconnected, it may still indicate that a log has been deleted in the middle.

There are two troubles with IoT logs: one is that verification cannot be started until the accident is over, and the other is that the gateway or platform cannot be trusted forever by default. Verification evidence should be kept synchronously during log generation, batch sealing, and cross domain transmission, otherwise database snapshots can only be relied upon afterwards. Low power terminals can reduce heavy calculations and hand over batch hashing to the gateway; Key control devices should retain local counters, trusted times, or device certificates, and bind log summaries with device status.

B. TECHNICAL LOGIC OF DETERMINISTIC MERKLE TREE

The so-called deterministic Merkle tree is not just about hashing the leaves all the way to the root, but about fixing all the rules that may cause ambiguity first. If there is no uniformity in the order in which leaves enter the tree, how fields are encoded, how null values are represented, how duplicate records are handled, and how odd nodes rise, different validators may calculate different root hashes even if they receive the same batch of logs. For the multi vendor, multi language, and multi platform environment of the Internet of Things, deterministic rules themselves are part of the security boundary.

At least three types of details need to be addressed during construction. Firstly, the original log should be converted into a stable key value object, with no ambiguity in character encoding, time format, numerical precision, Boolean values, and null values; The hash standard can detect whether the input bits have changed [6], but it does not help engineers determine whether the two writing methods are semantically equivalent. Secondly, leaf nodes and internal nodes should use different prefixes, such as 'leaf' and 'node', to avoid concatenation ambiguity. Thirdly, sorting, odd node promotion, and batch windows should be written as version rules, so that subsequent audits can recalculate the tree at that time.

C. EVIDENCE CHAIN REQUIREMENTS FOR TAMPERING TRACEABILITY

The biggest fear during the traceability stage is that there will only be one sentence: 'Verification failed'. What investigators really need to know is: which log has changed, whether the change is more likely to occur on the device, gateway, or platform, whether there have been rewrites or replays, whether the batch root has been properly anchored, and whether the tree headers seen by audit nodes are consistent. The W3C PROV data model describes data source relationships using entities, activities, and proxies [7], which can be translated into a log evidence chain:

original records, normalized records, hash nodes, batch roots, anchor credentials, and validation reports should all correspond to each other.

When discussing event response, the NIST Digital Evidence Guidelines place special emphasis on the reliability of evidence sources, collection processes, and processing methods [8]. In the context of this article, it is not enough to save a separate log text. It is also necessary to leave the correspondence between the original log and the normalized log, leaf hash and audit path, batch root hash and its anchoring records, as well as comparative explanations generated when verification fails [7]. Investigators can only distinguish between normal archive damage, changes caused by desensitization, and genuine malicious tampering by examining these materials together.

This article divides the overall framework into six core objects, as shown in Table 1 below.

TABLE 1. Core Objects in the Proposed Framework

Object	Description	Trust Function
Raw log	Original event generated by device, gateway, or platform	Preserve operational context
Canonical log	Normalized and deterministic representation of the raw log	Provide stable hash input
Leaf hash	Digest of canonical log with domain prefix and metadata	Bind content and source
Batch root	Merkle root generated within a fixed time or sequence window	Compress verification target
Anchor record	Signed or externalized root with timestamp and signer identity	Support independent audit
Trace report	Verification result with path, mismatch position, and responsible segment	Support tamper traceback

III. OVERALL FRAMEWORK DESIGN

A. DESIGN OBJECTIVES

The framework of this article prioritizes five things. The terminal side should be lightweight and should not allow low-power devices to frequently execute complex signatures; The construction results should be determined, and the same batch of inputs should receive the same root from different auditing parties; Historical batches should maintain an additional relationship and cannot be silently rewritten by the platform; The verification failure should be able to continue to move downwards, rather than stopping at the root hash inconsistency; Finally, the technical process should be able to embed key management, access control, and event response systems, otherwise it is difficult to adhere to them in real operations [8].

Therefore, the framework does not place all trust on a central database or chain, but adopts hierarchical collaboration. The device generates numbered logs, the gateway is responsible for batch encapsulation and tree construction, the platform saves indexes, paths, and root registrations, and audit nodes regularly sample and recalculate. When the security level is high, batch roots can be written to trusted timestamp services, consortium chains, or simply appended to object storage. Even if platform administrators can modify the repository, it is difficult to simultaneously change the snapshots held by external anchors and audit nodes.

B. SYSTEM ARCHITECTURE

In the deployment envisioned in this article, logs will not be directly thrown from the terminal into a central repository and considered complete. The device layer first records the sensor status, control commands, authentication behavior, configuration changes, and abnormal alarms, and includes basic fields such as Device-ID, Event_Seq, Event_Time, LogTypeType, Firmware_Vversion, etc. The edge layer is responsible for the first round of cleaning and sorting, checking for duplicates, missing fields, and device certificate status, and then generating Merkle trees by batch. The platform layer stores the log body, standardized copies, audit paths, batch roots, and anchor indexes, and provides query interfaces. The audit layer regularly retrieves the root hash and sampling proof, and if necessary, cross compares the tree heads seen by different nodes.

This arrangement is consistent with the zero trust mindset. The zero trust architecture emphasizes that access and data cannot be assumed to be trustworthy just because the device is on the intranet and the asset belongs to the same unit [9], [10]. In the framework of this article, both the gateway and platform need to be validated: batch roots need to be externally anchored, log paths need to be recalculated, and device serial numbers and certificate status need to be checked. If the platform database is modified, external roots and audit snapshots will expose differences; If the device forges logs, the serial number, firmware version, or certificate chain may also experience anomalies first. The responsibilities of each layer of the framework are shown in Table 2.

TABLE 2. System Layers and Operational Responsibilities

Layer	Main Responsibility	Key Output
Device layer	Generate event logs, maintain sequence, attach identity metadata	Raw log and local leaf digest
Edge layer	Validate schema, sort logs, build deterministic Merkle tree	Batch root and audit path
Platform layer	Store logs, index proofs, manage anchors, provide verification API	Root registry and query service
Audit layer	Perform inclusion proof, consistency proof, and cross-node sampling	Verification report
Governance layer	Define policies, retention rules, key rotation, and incident response	Control baseline

C. OPERATION PROCESS

A complete run can be understood as follows: the device first writes the event as the original log and provides an increasing sequence number; After receiving, the gateway supplements the receiving time, network source, and certificate status, and then converts the logs into standardized objects; Edge nodes collect a batch of leaves according to a preset window, generate root hashes, and audit paths for each log; Platform registration batch number, log quantity, root hash, and signature information; Audit nodes extract

several batches for recalculation at regular intervals. If a path cannot be traced back to its original root after recalculation, the system will then enter the traceability process based on the level of difference [11].

We need to separate the log text and supporting materials here. The main text often includes device location, control parameters, user identification, or business status, which can usually only be kept within the organization; The proof materials mainly include hash, path, root, and timestamp, which can be submitted to the auditing party for review with less risk. For example, in the investigation of vehicle road collaborative accidents, different units may not be willing to share the complete original logs, but they can first exchange batch roots, leaf hashes, and audit paths to confirm whether the same event exists, and then decide whether to retrieve more sensitive content [12].

IV. STANDARDIZATION OF LOGS AND CONSTRUCTION OF DETERMINISTIC TREES

A. LOG COLLECTION AND FIELD SPECIFICATION

Standardized fields should not be designed to be too full or too arbitrary. The minimum set for entering the hash should include device identification, log type, event time, receive time, sequence number, severity level, event subject, action, result status, and original payload summary. Business systems can add sensor values, geographic locations, control parameters, or work order numbers, but extended fields must be placed in a clear namespace. The time is in UTC millisecond format, with fixed numerical precision. The string is in UTF-8 format, and null values are explicitly written as null; These seemingly trivial rules determine whether they can be recalculated in the future.

During the collection phase, efforts should be made to generate non fallback serial numbers for the devices, which should be viewed together with the startup epoch, firmware version, and configuration version. After the device is powered off and restarted, the serial number can enter a new epoch, but cannot quietly return to the old range. When receiving through the gateway, the receiving time, network source, device certificate status, and duplicate detection results should be recorded to form a corresponding relationship between the original device records and the gateway receiving records. If field errors are found on the platform side, error correction instructions should be added instead of directly covering the standardized logs. The standardized log fields suggested in this article are shown in Table 3 below.

TABLE 3. Canonical Log Record Schema

Field	Type	Normalization Rule	Purpose
device_id	string	Lowercase identifier with namespace prefix	Bind source device
event_seq	integer	Monotonic value within device epoch	Detect deletion and replay
event_time	timestamp	UTC millisecond format	Preserve event order
receive_time	timestamp	Gateway observed UTC time	Measure delay and drift
event_type	string	Controlled vocabulary	Support classification

B. DETERMINISTIC SERIALIZATION

Deterministic serialization solves the problem of 'what byte string should be used to compute hash for the same object'. The JSON normalization scheme proposed in RFC 8785 states that a stable expression must be obtained before performing cryptographic operations [14]. IoT logs can learn from this principle: fields are arranged in lexicographic order, undefined fields do not participate in hashing, floating-point numbers are converted to fixed precision strings, arrays are only kept in their original order when they are ordered in business, otherwise they are sorted by element summary first. Sensitive binary payloads do not directly enter the object, but are generated as payload_ hash.

Merely specifying field sorting is not enough, the system also needs to indicate the schema_id for each log. The validator first selects the parsing rule based on the schema_id, and then performs serialization and hashing; This way, even if the fields are added in subsequent versions, the old logs will not be reinterpreted into another format. When the device or gateway temporarily does not support new fields, clear null values should be written or default values from version rules should be used, and the receiver should not be allowed to guess temporarily. Otherwise, a regular field upgrade may be misjudged as log tampering.

C. TREE CONSTRUCTION RULES

Building trees in batches is more suitable for the Internet of Things. A batch is defined by batch_id, Device_Scope, start_deq, end_deq, start_time, and end_time. After collecting all the logs in the window, the edge nodes sort them by device id, event_deq, event_time, and leaf_hash; Leaf computing uses the prefix "IOT-LOG-LEAF", while internal nodes use the prefix "IOT-LOG-MODE". When the number of nodes in a certain layer is odd, this article adopts the approach of raising the last node instead of copying nodes to avoid unnecessary duplicate explanations in the audit path [15].

After generating the root hash, the edge nodes also need to form tree_ head. This object records at least batch_id, root_hash, leaf_count, first_leaf, last_leaf, hash_alg, build_rule_version, builder_id And build_time. After signing the treehead, submit it to the platform, and the platform will generate an anchoring record. External storage only needs to save the tree_ head summary, without the need

to save the log body; After obtaining the same batch of logs and rules, the auditing party can recalculate and check whether the batch has been replaced, split, or merged. The construction rules for deterministic Merkle trees are shown in Table 4.

TABLE 4. Deterministic Merkle Tree Construction Rules

Rule Item	Required Decision	Security Effect
Leaf ordering	Sort by device_id, event_seq, event_time, and leaf_hash	Prevent order manipulation
Leaf prefix	Use fixed prefix IOT-LOG-LEAF	Separate leaf from internal node
Node prefix	Use fixed prefix IOT-LOG-NODE	Prevent concatenation ambiguity
Odd node handling	Promote the last node to upper layer	Avoid artificial duplication
Batch boundary	Fix time window and sequence range before hashing	Prevent silent batch rewriting
Tree head	Sign root, leaf_count, rule version, and builder identity	Support independent verification

V. INTEGRITY VERIFICATION MECHANISM

A. INCLUSION VERIFICATION OF A SINGLE LOG

The inclusion check answers a simple question: whether this log was actually included in this batch at that time. The validator first normalizes the log according to schema_id and calculates the leaf_hash, then obtains the path of the sibling nodes of the leaf, and recalculates the candidate roots from the leaf all the way up. If the candidate root is consistent with tree_ head and external anchor records, it indicates that the log already exists during batch sealing; If there is inconsistency, it is necessary to continue to determine whether it is the log body, path node, or batch root that has passed through [16].

The advantage of inclusive proof is that the length of the proof grows very slowly. A batch containing 65536 logs only requires approximately 16 sibling hashes to prove the location of a record. For gateways and audit links with limited bandwidth, this is much more realistic than transmitting entire batches of logs. To prevent attackers from forging paths, the path returned by the platform should also include the request time, source identifier, and server signature; High value events such as control commands, firmware upgrades, and administrator login can be self checked immediately after being stored [17].

B. BATCH CONSISTENCY VERIFICATION

Consistency verification focuses not on individual logs, but on whether the entire history has been rewritten. The certificate transparency system uses a small number of nodes to prove that the larger tree continues the smaller tree; IoT logs can also adopt a similar approach. If the system builds trees in a fixed batch, it can make the tree_cead of batch_n contain the root_cash or head_cash of batch_{n-1}. In this way, deleting, inserting, or rearranging a certain historical batch will affect the subsequent summary chain.

This mechanism is particularly suitable for discovering batch replacements. If an attacker only changes one record, inclusion verification can usually locate it; If he redo the entire batch, the batch may still appear consistent internally, but external anchors, batch chains, and audit node snapshots may not match. Therefore, the system should retain at least three references: tree_ head in the platform main library, head_ ash in external media, and head_ snapshot periodically pulled by audit nodes. If there are any differences among the three, they should be investigated at the batch level [18].

C. CROSS NODE REVIEW

Cross node review is used to prevent single point failures of platforms or gateways. Audit nodes can randomly check devices, time windows, and event types based on risk weights, not only checking whether the logs are in the tree, but also checking whether the device serial numbers are continuous, whether the receiving time is abnormal, whether multiple versions of the same event occur, and whether the batch size suddenly decreases. Key devices can also save the most recent leaf hashes locally; If the platform lacks corresponding records while the device side summary is still present, the problem is more likely to occur in the transmission link or platform storage. The integrity verification process is shown in Table 5 below.

TABLE 5. Integrity Verification Workflow

Step	Input	Operation	Expected Result
Schema check	Raw log and schema_id	Validate required fields and canonical rules	Canonical log is reproducible
Leaf recomputation	Canonical log	Calculate domain-separated leaf hash	Leaf hash matches index
Path verification	Leaf hash and audit path	Rebuild root from sibling nodes	Candidate root is obtained
Anchor comparison	Candidate root and anchored tree head	Compare root, leaf_count, and signature	Batch integrity is confirmed
Sequence review	Device sequence and previous leaf	Check gap, replay, and rollback	Temporal integrity is assessed
Report generation	All verification artifacts	Produce signed verification report	Evidence is available for audit

VI. TAMPERING TRACEABILITY AND APPLICATION DEPLOYMENT

A. IDENTIFICATION OF TAMPERING TYPES

From the verification results, tampering rarely takes on only one form. When a field is rewritten, the first thing exposed is a mismatch in leaf hash; When a log is deleted, there is often a gap in the device serial number, pre v-leaf, or leaf_count; Forged insertions are usually accompanied by unknown serial numbers, abnormal device identities, or irregular schema_ids; When the order is adjusted, batch boundaries and sorting rules will have issues first; Replay attacks manifest as repeated occurrences of the same leaf_hash within an unreasonable time window; If the platform tree_cead is not consistent with the external anchor, the problem may have escalated to root replacement or platform view splitting [19].

When determining the boundary of responsibility, materials from devices, gateways, platforms, and audit nodes should be viewed together. When the local serial number of the device is continuous and there is a summary on the device side but the platform is missing logs, the risk

is more inclined towards the gateway to platform link or platform storage; When the device serial number is rolled back and the firmware version, boot epoch, and time source are all abnormal, the possibility of the device being compromised is higher; If the platform root is consistent with the external anchor but the main text cannot be recalculated, it is necessary to first investigate the archive damage or desensitization process; When different audit nodes obtain different treeheads, attention should be paid to whether there is a split view in the log service.

B. TRACEABILITY PROCESS

When conducting actual investigations, it is not advisable to immediately flip through the entire log. A more prudent approach is to first check whether the platform root, external anchor root, and audit node snapshot are consistent, and determine whether the problem is batch level; If the roots are not consistent, then search down the audit path to find the sibling node where the difference first occurred; After locating the leaf, recalculate the standardized log to confirm whether the difference comes from the main text, path, or batch index; Finally, by combining device certificates, gateway reception records, administrator operation logs, and timestamp records, it is determined which entity or link is more likely to be held responsible [20].

The advantage of this tracing method is that it does not require retrieving all logs at once. Taking the batch of 32768 records as an example, when the root hash is inconsistent, the left and right subtree digests can be compared first, and then further downward positioning can be carried out. Usually, only a small number of path nodes are needed to narrow down the range to a specific leaf or a group of adjacent leaves. If the anomalies are concentrated on the same device, gateway, or maintenance period, the investigation direction will be significantly narrowed; If the anomaly crosses devices but is concentrated in the platform operation window, priority should be given to checking database scripts, index reconstruction, and administrator operation records. The tampering traceability evidence matrix is shown in Table 6 below.

TABLE 6. Tamper Traceback Evidence Matrix

Tamper Type	Observable Signal	Primary Evidence	Traceback Target
Content rewrite	Leaf hash mismatch	Canonical log, leaf hash, audit path	Field or payload handler
Log deletion	Sequence gap or leaf_count reduction	Device sequence, batch head, previous leaf	Device, gateway, or platform
Log insertion	Unknown sequence or invalid device identity	Device certificate and schema validation result	Compromised gateway or account
Replay attack	Duplicate leaf across abnormal window	Timestamp, epoch, and receive record	Network link or collector
Root replacement	Anchored root differs from platform root	Anchor record and signed tree head	Platform storage or administrator
Split-view attack	Auditors receive inconsistent tree heads	Audit snapshots and query signatures	Log service or gateway proxy

C. DEPLOYMENT STRATEGY

It is not necessary to use the same strength for all devices during deployment. Ordinary environmental collectors can place the main calculations at the gateway, while the terminal only maintains the serial number and identity fields; It is best for production control equipment to generate leaf hashes on the device side and sign key log summaries; Equipment involving personal safety, fund settlement, or regulatory auditing should be equipped with independent audit nodes and external timestamp services. The focus of grading is not on whether or not to use Merkle trees, but on where the leaves are generated, how often the roots are anchored, how high the sampling rate is, and how long the evidence is retained.

The performance overhead mainly occurs in four locations: standardized serialization, hash calculation, path saving, and external anchoring. Edge nodes can compute leaves and upper level nodes in parallel, and paths may not necessarily be fully unfolded in advance. Common batches can be cached, and cold data can be generated on demand.

High frequency equipment is suitable for forming a batch of one minute or a fixed number of pieces, while low-frequency equipment can be aggregated by hour or by event type. Anchoring too slowly can elongate the tampering exposure window, while anchoring too frequently can increase costs. Therefore, anchoring critical events in real-time and regular running logs at regular intervals is more in line with engineering habits.

D. GOVERNANCE GUARANTEE

Without institutional support, technical solutions are easily stuck in the demonstration stage. Organizations need to first classify and grade logs, clarify which fields enter the hash, which sensitive fields only retain the summary, and which records must be retained for a long time; Secondly, it is necessary to manage device keys, gateway signature keys, and audit node keys to avoid one account having the authority to generate, delete, and modify anchors simultaneously; It is also necessary to include access control, event response, and system integrity requirements in the operation and maintenance procedures. The audit, access control, and event response control families in the NIST Security and Privacy Control Catalog can serve as mapping criteria.

Long term stored log evidence may also encounter algorithm lifecycle issues. For systems with high signature load and long evidence retention period, we can refer to the suggestion of stateful hash signature to evaluate resistance to quantum migration and key state management. Crosby et al.'s research on tamper proof log structures also indicates that a well-designed digest structure is necessary to strike a balance between storage overhead, verification efficiency, and audit interpretability. This means that the framework should allow algorithm upgrades and rule version migration in the future, rather than binding all batches to a one-time design.

Privacy constraints should also be written into the design in advance. IoT logs may contain location trajectories, personnel behavior, health status, process parameters, or trade secrets, and integrity protection cannot become a new channel for leakage. A more secure approach is to keep the log body internally and only save the summary

and anchoring results externally; When a third party needs to prove the existence of a record, they should prioritize providing the leaf hash, audit path, and batch root instead of directly handing over the original log; When collaborating across institutions, include the retention period, access purpose, desensitization criteria, and responsible parties in the process.

VII. CONCLUSION

This article discusses a more specific engineering problem: how to prove that IoT logs have not been deleted, edited, supplemented, or rearranged after they flow between devices, gateways, and platforms. The answer given in the article is not to rely solely on a central repository, nor to fully chain the logs, but to use a deterministic Merkle tree to connect the log body, hash path, batch root, and external anchor. As long as standardized rules, tree construction rules, and audit paths can be recalculated, log integrity can shift from "trusted by administrators" to "verifiable by auditors".

From an application perspective, the value of a framework is not just about discovering a hash inconsistency, but more importantly, explaining the inconsistency clearly: which batch it belongs to, which layer the path is broken at, which log the leaf corresponds to, and whether field changes can be traced back to the device, gateway, platform, or operations account number. Subsequent research can continue with three things: integrating secure elements and trusted execution environments into the leaf generation process; Incremental proof in compressed ultra large scale terminal scenarios; Link the verification report with work orders, alarms, and automated disposal systems to truly integrate log verification into daily operations.

ACKNOWLEDGMENT

This work was supported by Natural Science Research Project for Anhui Universities (KJ2018A0336, KJ2021A0682, 2022AH051324, 2023AH050403, 2023AH050406, 2024AH051466), Fuyang Normal University Open Project of Anhui Intelligent Computing and Information Innovation Application Engineering Research Center (ICII202307), Scientific Research Project of Fuyang Normal University(Analysis of Security Protocol for IoT Perception Layer Based on RFID), Fuyang Normal University Quality Engineering Project (2023JYXMSZ09, 2024XTTZTD03), Anhui Province Quality Engineering Project (2022jyxm1172, 2023jyxm0528, 2023yjshhsfkc036).

REFERENCES

[1] B. Vrooman, "Optimizing Merkle Tree Operations in a High-performance Blockchain Virtual Machine: Architectural Approaches and Performance Analysis," Asian Journal of Research in Computer Science, vol. 19, no. 4, pp. 165–176, 2026, doi: 10.9734/AJRCOS/2026/V19I4856.

[2] O. Hammoud and A. I. Tarkhanov, "Correction: A scaling distributed access control model for blockchain-based file storage systems," Frontiers in Blockchain, vol. 9, pp. 1806905–1806905, 2026, doi: 10.3389/FBLOC.2026.1806905.

[3] A. Corsi, A. Almenhali, and N. Laurenti, "Analysis of collision probability in Merkle trees in the random oracle model," Journal of Mathematical Cryptology, vol. 20, no. 1, pp. 20250050–20250050, 2026, doi: 10.1515/JMC-2025-0050.

[4] S. Arunadevi and P. Valarmathie, "Privacy-Preserving and Scalable Electronic Health Record Management Using Multi-Layer Merkle Trees and Zero-Knowledge Proofs," International Journal of Pattern Recognition and Artificial Intelligence, vol. 40, no. 5, 2026, doi: 10.1142/S0218001425590177.

[5] M. Zheng, S. Huang, D. Kong, et al., "Logarithmic-Size Post-Quantum Linkable Ring Signatures Based on Aggregation Operations," Entropy, vol. 28, no. 1, pp. 130–130, 2026, doi: 10.3390/E28010130.

[6] M. A. M. M. Alsaedy, Z. A. Ghalwash, E. A. A. Yousif, et al., "E-AAPIV: Merkle Tree-Based Real-Time Android Manifest Integrity Verification for Mobile Payment Security," Journal of Cyber Security, vol. 7, no. 1, pp. 653–674, 2025, doi: 10.32604/JCS.2025.073547.

[7] S. Kodadi, K. Dondapati, D. P. D. Deevi, et al., "Optimizing Supply Chain Finance with XGBOOST and Merkle Tree Blockchain," International Journal of Innovation and Technology Management, vol. 22, no. 7–08, 2025, doi: 10.1142/S0219877025400061.

[8] R. Du, Z. Wang, and J. Shen, "Certificateless data integrity auditing with sparse Merkle trees for the cloud-edge environment," Scientific Reports, vol. 15, no. 1, pp. 39202–39202, 2025, doi: 10.1038/S41598-025-14041-9.

[9] M. S. Prabhu, N. Subramanyam, P. S. Krishnan, et al., "Decentralized digital currency system using Merkle hash trees," Journal of Banking and Financial Technology, vol. 9, no. 2, pp. 1–31, 2025, doi: 10.1007/S42786-025-00059-0.

[10] A. V. H. Le, N. D. Q. Nguyen, N. Tadashi, et al., "Blockchain-Based Decentralized Identity Management System with AI and Merkle Trees," Computers, vol. 14, no. 7, pp. 289–289, 2025, doi: 10.3390/COMPUTERS14070289.

[11] G. Liu, H. Lu, W. Wang, et al., "An efficient authentication scheme for vehicular networks based on Merkle tree," Computer Networks, vol. 269, pp. 111429–111429, 2025, doi: 10.1016/J.COMNET.2025.111429.

[12] P. Liu, D. Luo, J. Liu, et al., "CMT-YARN: an efficient security framework for yarn based on an improved merkle tree," The Journal of Supercomputing, vol. 81, no. 10, pp. 1098–1098, 2025, doi: 10.1007/S11227-025-07571-6.

[13] S. Narla, S. Peddi, T. D. Valivarthi, et al., "FOG computing based energy efficient and secured iot data sharing using SGSOA and GMCC," Sustainable Computing: Informatics and Systems, vol. 46, pp. 101109–101109, 2025, doi: 10.1016/J.SUSCOM.2025.101109.

[14] Y. Wu, T. Feng, C. Su, et al., "MSAUPL: A multi-server authentication and key agreement protocol for industrial IoT based on user privacy level," Journal of Information Security and Applications, vol. 89, pp. 103991–103991, 2025, doi: 10.1016/J.JISA.2025.103991.

[15] S. Liu, X. Zhou, A. X. Wang, et al., "A hash-based post-quantum ring signature scheme for the Internet of Vehicles," Journal of Systems Architecture, vol. 160, pp. 103345–103345, 2025, doi: 10.1016/J.SYSARC.2025.103345.

[16] S. S. Fateminasab, D. Bahrepour, and K. R. S. Tabbakh, "A fair non-collateral consensus protocol based on Merkle tree for hierarchical IoT blockchain," Scientific Reports, vol. 15, no. 1, pp. 3645–3645, 2025, doi: 10.1038/S41598-025-87025-4.

[17] Y. Meng, B. Wang, Q. Xing, et al., "BBAD: Blockchain-based data assured deletion and access control system for IoT," Peer-to-Peer Networking and Applications, vol. 18, no. 2, pp. 1–1, 2024, doi: 10.1007/S12083-024-01881-X.

[18] M. Nasreen and K. S. Singh, "BPMT: A hybrid model for secure and effective electronic medical record management system," Journal of Computational Science, vol. 83, pp. 102457–102457, 2024, doi: 10.1016/J.JOCS.2024.102457.

[19] E. Mollakuqe, H. Dag, and V. Dimitrova, "Mathematical Foundations and Implementation of CONIKS Key Transparency," Applied Sciences, vol. 14, no. 21, pp. 9725–9725, 2024, doi: 10.3390/APP14219725.

[20] A. Romero and R. Hernandez, "Blockchain-Driven Generalization of Policy Management for Multiproduct Insurance Companies," Future Internet, vol. 16, no. 10, pp. 356–356, 2024, doi: 10.3390/FI16100356.