# Integrity Completeness for Supply Chain Data: Trust Model Analysis and EVM Realization

Roman Sytnyk[1,*] and Viktoriia Hnatushenko[1,2]

[1] Ukrainian State University of Science and Technology, Lazaryana str. 2, Dnipro 49000, Ukraine
[2] Dmytro Motornyi Tavria State Agrotechnological University, Universitetskaya St., 66, Zaporizhzhia, Ukraine, 69000

Abstract

Shared supply chain data can be silently modified or deleted by an adversary who controls the data store. This paper formalizes the property of integrity completeness against such an adversary and compares three trust models under this threat: symmetric hashing with HMAC chains (Model A), per-record ECDSA signatures (Model B), and blockchain with application-level Merkle trees (Model C). Only Model C achieves integrity completeness, because record deletions cannot be detected under Models A and B. A four-contract EVM architecture is proposed to realise Model C: it combines Merkle tree construction, role-based access control, state mutation and audit logging in a single transaction, so the EVM revert semantics bind every integrity proof to its state change. The system was evaluated on synthetic datasets of 1,000, 10,000 and 100,000 records. Full verification scales linearly with $\alpha$ between 0.99 and 1.05. In practice this means an organisation can detect silent record deletions, which both symmetric hashing and per-record signatures miss, while paying less for verification than per-record ECDSA. It keeps trust-minimised integrity practical for supply chain batches on blockchain networks.

Keywords

Trust Models, Merkle Trees, Smart Contracts, Data Integrity, Blockchain, EVM, Supply Chain

## 1. Introduction

Modern supply chains include many independent organizations which exchange data about resources, products and shipments [1, 2, 3]. When such networks grow, it becomes important to decide who is allowed to keep and manage the shared data. This question affects the design of the whole system, not only its day-to-day operations. Companies usually have two options here: they can use one operator who hosts all the data for the network, or they can keep synchronized copies of the data on their own side. Both options require some level of trust between participants, and in real situations this trust often does not exist. Also, cyber-attacks on supply chains usually target exactly these inter-organizational boundaries [4].

Integrity verification can miss two kinds of tampering. It can fail to reject a modified or forged record, and it can fail to notice a deleted one. We say a verifier has integrity completeness when it misses neither, and accepts the store only when it holds exactly the honest dataset. Deletion is the harder case once the data store itself is adversarial. The scientific contribution of this paper is a comparative analysis of integrity completeness under such an adversary. We define it for a multi-party supply chain (Definition 1), and we prove which classes of integrity mechanisms satisfy it and which do not (Proposition 1).

We study three classes: symmetric hashing with HMAC chains, per-record ECDSA signatures, and blockchain with application-level Merkle trees. Proposition 1 is a separation result. Symmetric hashing cannot satisfy completeness, because the administrator holds the key and recomputes valid hashes. Per-record signatures cannot satisfy it either, because the verification function checks only the records that are present and never the size of the set, so a deletion stays invisible. We also show

[1]RS-2026: Resilient Systems: Secure Digital Technologies and Critical Infrastructure, June 30, 2026, Drohobych, Ukraine
* Corresponding author.
r.sytnyk@outlook.com (R. Sytnyk), vvitagnat@gmail.com (V. Hnatushenko)
0000-0001-7820-9128 (R. Sytnyk); 0000-0001-5304-4144 (V. Hnatushenko)

that the usual repairs of the signature model, such as chained signatures or a periodically signed root, do not close this gap but move the trust to another party. Only the third class satisfies completeness, and the analysis explains why: completeness needs a tamper-evident commitment to the whole record set, kept in a place the administrator cannot rewrite. This condition decides the property, and not the choice between hashing and signatures.

We then turn the positive case into a concrete system and use it to measure the real cost of the only model that satisfies completeness. We propose a four-contract EVM architecture that builds the application-level Merkle tree over supply chain metadata and binds it to role-based access control, state changes and audit logging in one transaction, so that a state change is committed only together with its integrity proof. We evaluate it on synthetic datasets up to 100,000 records and on two public testnets.

## 2. Relevance of the Research

There are several technical solutions which try to solve this problem. Inside a single organization, relational databases with checksums and HMAC chains can protect data when there is a trusted database administrator. In regulatory and financial systems, per-record ECDSA signatures are used to provide non-repudiation of individual records. Blockchain platforms such as Ethereum and Hyperledger Fabric [5, 6] use protocol-level Merkle trees (transactionsRoot, stateRoot, receiptsRoot) for proving that a transaction is included in a block. Supply chain platforms like VeChain and IBM Food Trust are built on top of these guarantees. Smart-contract libraries such as OpenZeppelin's MerkleProof [7] and the on-chain MerkleTree library [8] also provide on-chain proof verification, which is often used in decentralized finance (DeFi) for airdrops and token distribution.

None of these approaches solves the full problem in one architecture. Database checksums do not help against a malicious administrator, who holds the symmetric key and can recompute valid hashes for modified records. Per-record ECDSA catches modifications, but it cannot catch deletions, because the verification function checks only the records that still remain in the database. Protocol-level Merkle trees in Ethereum prove that a transaction is included in a block, but they do not say anything about whether the business data inside this transaction is correct [9]. Existing supply chain platforms rely on protocol immutability and do not build application-level Merkle trees over business records [1, 2, 6, 10]. Recent surveys of smart-contract vulnerabilities also show that combining these subsystems is an error-prone task [11].

Several systematic reviews and mapping studies have already surveyed this field [1, 2, 3, 10]. All of them agree on the same point: integrity verification, traceability and trust between participants are the main reasons why companies adopt blockchain in supply chains. The 2025 review [10] proposes a framework for evaluating security, traceability and data integrity. It also notes that most existing implementations rely on protocol-level immutability and do not build cryptographic structures at the application level. A focused review of food supply chains [3] analyses 80 papers and finds the same pattern: conceptual proposals are common, but measured performance results are rare.

Real-world platforms like VeChain, IBM Food Trust and Hyperledger Fabric [6] are built on the protocol-level Merkle trees of the underlying ledger, and they cannot verify a whole batch of business records in one operation. Our earlier work [12, 13] is the starting point for this paper, but it did not include a full EVM implementation with measured performance against ECDSA and SHA-256 baselines. On-chain Merkle libraries [7, 8] are a separate strand of work. OpenZeppelin's MerkleProof library combines off-chain tree construction with on-chain proof verification. It is widely used today for airdrops and token distribution. However, neither of them provides leaf encoding for supply chain metadata, and neither connects proof verification with role-based access control, state management and event logging in the same transaction. The quad-Merkle audit scheme [14] shows that on-chain auditing in place of third-party auditors is workable, although its leaves are generic data blocks, not structured supply chain events. More recent work on updatable Merkle batch proofs [15] also uses the standard sorted-pair construction, but in order to support efficient updates over

moving streams of blocks. It does not address domain-specific encoding for supply chain metadata as well.

Smart-contract security is the third relevant topic. The survey of Ethereum attacks [16] identified the main attack classes. The recent systematic analysis [11] covers six categories of vulnerabilities and confirms that static analysis with tools such as Slither [17] is now the standard practice for checking contract correctness. IPFS [18] also uses Merkle structures, but at the file and block level, not over structured business metadata. So it does not directly answer the supply chain integrity question.

A few more sources support deployment feasibility. Arbitrum Nitro [19] is a second-generation optimistic rollup that runs the same EVM bytecode at much lower transaction cost. The BRUINchain pilot [20] under the US DSCSA pharmaceutical regulation shows that blockchain-based last-mile verification works in practice. The Ethereum Yellow Paper [9] provides the formal EVM execution semantics on which the four-contract pattern used in this paper relies.

There are different solutions for integrity verification in supply chain information systems. However, in the published literature so far there is no unified architecture which combines application-level Merkle trees over supply chain metadata, atomic role-based access control and audit logging in one transaction. We build on this line of work and extend it in two directions. We set the adversary to be the data store administrator of a multi-party supply chain, and we give a separation result that shows which mechanism classes keep completeness under this adversary, and which ones only appear to.

## 3. Statement of the Problem

Let A be an adversary which works against a multi-party supply chain. The system has n participants $\{P_1, ..., P_n\}$ and a shared data store D. In the blockchain case (Model C) the system also has an additional consensus validator set V of size v, which may or may not overlap with the participant set. All three models assume that the underlying cryptographic primitives are secure under standard hardness assumptions. SHA-256 and keccak256 are collision-resistant, and secp256k1 ECDSA cannot be forged by an attacker who only sees signed messages (the EUF-CMA assumption). The cryptography is the same across all three models. The difference between the models is in how much of the system A controls.

In Model A (symmetric hashing) A controls any party with write access to D, including the database administrator. A can read, modify, insert and delete records. Also, A can recompute any symmetric hash or HMAC, because the shared key is available to anyone who has write access.

In Model B (per-record ECDSA) A controls the database administrator, but does not hold any honest participant's private key. A can read all records, delete records and modify unsigned metadata, but it cannot forge a valid ECDSA signature for any honest participant. Other powers of A are the same as in Model A.

In Model C (blockchain + Merkle trees) A controls up to $f < v/3$ of the v consensus validators in a BFT network, or strictly less than 50% of the total hashpower in a PoW network. Once a record is committed to the ledger, A cannot modify, delete or insert it alone. Invalid transactions from A are rejected by on-chain access control and by the consensus rules.

The security goal in all three models is integrity completeness, which we state as a property of the verification function: it accepts the store only when the data equals the honest dataset.

Definition 1 (Integrity completeness). Let $D = \{r_1, ..., r_m\}$ be the honest dataset. Let V be the verification function, which reads the current store and answers accept or reject. We use a simple tampering game. The adversary A starts from the honest store for D and changes it, using only what its model allows, into a store that holds a dataset D'. A wins the game if D' is different from D, by at least one added, deleted or changed record, and V still answers accept. The mechanism has integrity completeness against A if A can win only with negligible probability in the security parameter $\lambda$. In other words, V answers accept only when D' = D, except with negligible probability.

This is a standard integrity goal for an outsourced store: the verifier accepts only the exact honest dataset, and any tampering is rejected. We study it here against an adversary that controls the store.

Whether this property holds depends on which adversary model we are in. The problem addressed in this paper is to determine which of the three trust models achieves integrity completeness under adversary A, and then to realize the model that does in a practical supply chain system.

# 4. Research Results

## 4.1. Trust models under the adversary

Proposition 1. Let A be a computationally bounded adversary which controls the data-store administrator. Under Model A (symmetric hashing with HMAC chains), the verification function V(D) cannot detect modifications by A, because A can recompute valid hashes using the shared key. Under Model B (per-record ECDSA), V(D) cannot detect deletions by A, because V(D) checks authenticity only over the records which remain in D. Under Model C (blockchain with application-level Merkle trees), V(D) detects both modifications and deletions by A, provided that the consensus assumption holds ($f < v/3$ for BFT, or strictly less than 50% adversarial hashpower for PoW).

Proof. We check the three models one by one, using the tampering game of Definition 1. In Model A, the verification data is a set of SHA-256 hashes and an HMAC chain under a shared key K, and the administrator has this key. So A can replace a record $r_i$ with r' and also write the correct hash and HMAC(K, r') for it. Then V accepts the store, even though D' is different from D, and A wins. Changes are not detected, so integrity completeness does not hold. In Model B, every record has an ECDSA signature from the party that created it, and A cannot forge such a signature (the EUF-CMA assumption). So a changed or added record is rejected, but a deleted record is not. Let D' be D without one record $r_i$. Each record that stays in D' still has its own valid signature, and V checks only the records that are present; there is no stored value that says how many records there should be. So V accepts D', although it is different from D, and A wins. Deletions are not detected, so integrity completeness does not hold. In Model C, the verification data is the Merkle root R, which is stored on-chain at every update. Under the consensus assumption ($f < v/3$ for BFT, or less than 50% of the hashpower for PoW), A cannot change R. To win, A would need a dataset different from D that still gives the same root R. But a change to a record changes its leaf hash (Eq. 1) and, through the sorted-pair rule (Eq. 2), changes the recomputed root; a deletion or an insertion changes the set of leaves and so changes the root too. Both cases need a keccak256 collision, which a bounded adversary finds only with negligible probability. So V accepts only when D' = D, except with negligible probability, and integrity completeness holds.

Table 1
Trust model comparison under adversary A.

| Threat | Model A: SHA-256 + HMAC | Model B: Per-Record ECDSA | Model C: Blockchain + Merkle |
|---|---|---|---|
| External tampering | Detected (hash check) | Detected (signature check) | Detected (root mismatch) |
| Record deletion | Not detected | Not detected | Detected (root mismatch) |
| Insider data replacement | Not detected (A recomputes hashes) | Detected (A cannot forge signatures) | Detected (consensus prevents) |
| Administrator collusion | Not mitigated | Partially mitigated (deletion risk) | Mitigated (breaks only if $f \geq v/3$) |
| Trusted party required | Yes (DB admin) | Yes (DB admin, partial) | Trust-minimized (consensus) |
| Full-dataset verification | O(n) symmetric | O(n) asymmetric | O(n) symmetric |
| Single-record verification | O(n) (full rescan) | O(1) (one signature) | O(log n) (Merkle proof) |

The three cases of Proposition 1 are discussed in more detail below.

### 4.1.1. Database with symmetric hashing (SHA-256 + HMAC chain)

Each record contains a SHA-256 hash of its content together with an HMAC that links it to the previous record. Because the administrator holds the symmetric key, A can change the data and recompute valid hashes without being detected. External tampering is still caught, at a cost of O(n) symmetric hash operations. Model A is good enough for single-organization systems which trust their internal operators. Integrity completeness holds only when A has no write access to D, and this is the weakest of the three assumptions.

### 4.1.2. Database with per-record ECDSA signatures

Each record is signed with the private key of the originating actor. Signatures block forgery, but this model does not achieve completeness. A malicious administrator can delete records without detection, because V(D) checks authenticity only of the records which are present in the database. It has no way to notice records which are missing. Per-record ECDSA verification is also computationally expensive, and this cost is part of the non-repudiation guarantee. Model B works for multi-party environments where participants do not trust the database operator, but can accept silent omissions of records.

There are also chained variants of Model B which try to narrow the deletion gap. In one variant, each record's signature covers its own content together with the previous record's signature. In another variant, the administrator periodically signs a Merkle root over all current records. Neither of these variants removes the trust dependency. The first one needs an honest aggregator to sign the chain tip, which brings back a trusted party. The second one still requires trusting the administrator who computes and signs the root, which is the same dependency that the model is supposed to avoid. In these variants the trust dependency is moved to another participant, but it is not actually removed. In Model C the consensus mechanism replaces any single participant in this role.

### 4.1.3. Blockchain with application-level Merkle trees

Data is distributed across a consensus network. An adversary which controls fewer than v/3 validators cannot modify or delete historical records alone. The Merkle root is a compact commitment to the dataset state. Any modification or deletion changes the root, so the change can be detected. Integrity completeness holds as long as the consensus assumption is satisfied. In Model C no single party holds the trust. The trust is moved to the consensus mechanism under its liveness and safety assumptions ($f < v/3$ for BFT, or strictly less than 50% adversarial hashpower for PoW). Some trust also remains in the cryptographic primitives and in the client software which verifies block headers and proofs. These assumptions are weaker than the ones in Models A and B. Verification uses symmetric hashing (keccak256) and avoids per-record asymmetric signatures, but it requires complex blockchain infrastructure and gas costs. Supply chain participants are usually independent organisations which do not trust each other, and this is the threat model that centralised alternatives cannot handle [2, 10].

## 4.2. Merkle tree at the business-logic level

The trust-model analysis above shows that only the application-level Merkle tree can detect both deletions and modifications when an adversary controls the data store. For the verification to be possible at the business level, the tree should be built over the structured metadata which participants of the supply chain exchange, and not over generic byte blocks. In the proposed system the tree is built directly inside a smart-contract. The integrity proofs are produced by the same code which updates the business state, and not by an external tool.

Each supply chain event generates a leaf node, which is computed as:

leaf = keccak256(abi.encodePacked(resourceId, locationData, block.timestamp, msg.sender)) (1)

Here resourceId is the uint256 identifier of the tracked asset, locationData encodes geographic coordinates or a warehouse identifier, block.timestamp binds the event to the time of the consensus-

validated block, and msg.sender is the Ethereum address of the transaction signer. The abi.encodePacked function returns a tightly packed byte representation without padding, which reduces gas consumption.

In Ethereum protocol-level Merkle trees the leaves are transaction objects. In the proposed system the leaves are logistics metadata, so the tree becomes a verifiable structure over business data. There is a known 64-byte pre-image attack, where a leaf is faked as the joined hash of two 32-byte internal nodes [7]. This attack does not apply here. The packed representation (uint256, bytes, uint256, address) is at least 84 bytes long even with empty locationData, which is well above the 64-byte limit. Encoding ambiguity is also not a problem: according to the Solidity ABI specification, abi.encodePacked becomes non-injective only when two variable-length parameters are placed next to each other, and locationData is the only such parameter in our encoding.

Tree construction uses the standard sorted-pair hashing rule, which is compatible with [7] and is resistant to second-preimage attacks:

$$\text{parent} = \text{keccak256}(\text{abi.encodePacked}(\min(\text{left}, \text{right}), \max(\text{left}, \text{right}))) \quad (2)$$

The tree is then built level by level, from the leaves up. At each level the nodes are taken in pairs, and each pair is hashed with the rule in Eq. (2) to form one parent node. If a level has an odd number of nodes, the last node is moved up to the next level without change. The process repeats until only one node is left; this node is the Merkle root. Each node is processed once, so the construction takes O(n) time, and no special case is needed when the number of leaves is not a power of two. The same sorted-pair construction is also used by recent on-chain Merkle proof systems [15] for blockchain applications.

A single record can be checked without rebuilding the whole tree, by using an inclusion proof. The proof is the list of sibling hashes along the path from the leaf to the root. The check starts from the leaf hash and combines it with each sibling in turn, using the same ordered rule, until it reaches a root value. If this value is equal to the stored root, the record belongs to the tree. The check follows only one path, so it takes O(log n) time. For n = 10,000 it needs only 14 hash steps, against 10,000 hashes for a full rebuild. A JavaScript implementation based on ethers.js mirrors the Solidity contract byte-for-byte. It uses the same sorted-pair ordering and the same keccak256 encoding via ethers.solidityPackedKeccak256. So Merkle roots can be cross-validated without deploying to a blockchain. This is useful for off-chain integration with traditional information systems.

## 4.3. Four-contract architecture

In this subsection the Merkle tree described above is combined with the rest of the supply chain logic. The aim is to produce the integrity proof together with the business operation in every case, instead of generating it as a separate optional step. In classical information systems, access control, business logic, integrity checking and audit logging usually live in separate services or modules. This separation can lead to inconsistencies under network failures or partial errors. For example, the data can be updated, but the corresponding integrity record is missing. Or the audit log can show a state different from the one stored in the database. The architecture described in this paper relies on EVM transaction-level atomicity to avoid these problems. The four subsystems are bound into a single transaction. If any of the four operations fails, the EVM reverts everything together [9].

The system uses four smart-contracts, one for each separate concern. This kind of separation by responsibility is similar to general software engineering practice, but it has to fit inside the EVM environment.

Table 2
Four-contract architecture.

| Contract | Responsibility | Key Functions |
|---|---|---|
| SupplyChainCore.sol | Order lifecycle, ownership state, production status | addOrder, updateProductionStatus, confirmReceipt |

| | | |
|---|---|---|
| AccessControl.sol | Role-based access: Supplier, Manufacturer, Transporter, Distributor | grantRole, checkRole, modifier enforcement |
| MerkleVerification.sol | Merkle tree construction, proof generation and verification | calculateLocationMerkleRoot, verifyMerkleProof, buildMerkleTree |
| MovementManager.sol | Immutable location and ownership transfer history | applyLocation, transferOwnership, getLocationHistory |

Each business operation passes through three contracts. There is an entry-point contract that manages the state, AccessControl for role checking, and MerkleVerification for proof updates. For example, SupplyChainCore.addOrder() first checks the Supplier role through AccessControl, then updates the order state, then calls MerkleVerification to append a new leaf. The two entry points are SupplyChainCore and MovementManager. They share the same AccessControl and MerkleVerification instances. Under EVM revert semantics, access control, state mutation and Merkle root updates are all bound to the same transaction. So in the proposed system a state change can be committed only together with the corresponding integrity proof.

The system has two layers of cryptographic protection. Every transaction is ECDSA-signed on secp256k1 [9], which provides transaction-level authenticity. The Merkle tree adds a second layer of protection on the dataset level. Any modification of a record invalidates the root, so the change can be detected just by comparing the roots. Both these protection layers are applied inside the same transaction.

For verification of the implementation, Slither v0.11.5 [17] was run with all 101 detectors against the four contracts. It reports zero high-severity and zero medium-severity findings. Static analysis with Slither is standard practice for checking EVM contracts [11]. It flags known vulnerability patterns, but it is not a proof of correctness. The full report is available in the supplementary material.

## 4.4. Experimental evaluation

### 4.4.1. Setup and methodology

The experiment compared three systems for supply chain data integrity. Each of them was chosen to test a different aspect of the cost.

The primary system is the four-contract architecture described above. It was deployed on a Hardhat development network (in-process EVM, auto-mining, gas limit 300 billion). It used Solidity v0.8.18 with the optimizer enabled at 200 runs and ethers.js v6 on the off-chain side.

The SHA-256-only baseline was built on Node.js v25.6.1 and MySQL v8.0, with a five-table schema which matches the four-contract architecture. Integrity here is verified by recomputing SHA-256 hashes and running multi-table JOINs. This setup measures hash-based tamper detection on its own, without consensus and without non-repudiation. It is also a lower bound for the cost of on-chain Merkle construction.

The Model B reference uses the same schema as the SHA-256 baseline, but adds per-record ECDSA verification. To separate the cost of the model itself from the cost of any single library, secp256k1 verification was run through two implementations. The first one is node:crypto, which is backed by OpenSSL. The second one is @noble/curves v1.9.7, an audited pure-TypeScript library which is also used by ethers v6 and viem. A cross-library check confirmed that signatures produced with elliptic v6.6.0 verify correctly under both implementations. Model B was measured at 1,000 and 10,000 records. The 100,000-record case was skipped, because the cost grows linearly and this was already confirmed at the 1k → 10k step.

All experiments were run on an Apple MacBook Pro M4 with 24 GB of unified memory. The MySQL container had an 8 GB InnoDB buffer pool, four CPU cores and an 8 GB memory limit. Three synthetic datasets were used (1,000, 10,000 and 100,000 records), each with 8–12 status history entries per order.

### 4.4.2. Full-dataset verification results

Table 3
Integrity verification time: blockchain Merkle construction vs. SHA-256-only baseline and per-record ECDSA (Model B). Mean of 10 runs, first 2 discarded as warmup. Baseline excludes consensus and non-repudiation costs.

| Dataset | MySQL (SHA-256 only) | Blockchain (Merkle) | MySQL + ECDSA (node:crypto) | MySQL + ECDSA (@noble) |
|---|---|---|---|---|
| 1,000 | 12.8 ms (σ=3.0 ms) | 92.8 ms (σ=8.8 ms) | 312 ms (σ=1 ms) | 1.95 s (σ=38 ms) |
| 10,000 | 59.0 ms (σ=2.0 ms) | 1.03 s (σ=55.4 ms) | 3.06 s (σ=40 ms) | 18.64 s (σ=66 ms) |
| 100,000 | 623 ms (σ=8.0 ms) | 10.06 s (σ=196.0 ms) | – (not measured) | – (not measured) |

Model C verification is 7-17 times slower than the SHA-256-only baseline. The overhead comes from EVM execution costs (per-leaf SSTOREs and recursive tree construction), because both paths perform the same O(n) symmetric hashing. The full-dataset construction in Table 3 measures the computational cost of building the whole tree at once. In deployment the tree grows by one leaf per event, so this is an upper bound and not a per-transaction cost; on a public network a 100,000-record build would be split across many blocks.

At 10,000 records, on-chain Merkle construction is 3.0 times faster than OpenSSL-backed node:crypto and 18.1 times faster than pure-TypeScript @noble/curves. A third pure-JS library, elliptic, takes 17.02 s, which is within 10% of @noble/curves. The 6.1× spread between optimized C and pure-TypeScript ports is expected. Both libraries verified all 21,986 records in the 10k dataset (10,000 location entries, 1,986 ownership transfers and 10,000 audit-log entries across the three MySQL tables). Per-record ECDSA verification is slower than full-dataset Merkle construction in Model C in every measured case. However, per-record verification is the worst case for Model B. Faster Model B variants exist, such as signed Merkle roots over all records, or batch-signature schemes. But they trade away part of the per-record non-repudiation guarantee, which is what makes Model B attractive in the first place.

### 4.4.3. Testnet validation

The same four-contract deployment was also tested on two public Ethereum testnets, Sepolia L1 and Arbitrum Sepolia L2, with the 1,000-record dataset. The aim was to check that the local Hardhat numbers also hold under real consensus and real network latency.

Table 4
Testnet validation (1,000 records, mean of 10 runs, first 2 discarded as warmup).

| Metric | Hardhat (local) | Sepolia L1 | Arbitrum Sepolia L2 |
|---|---|---|---|
| Off-chain verification | 92.8 ms (σ=8.8 ms) | 91.8 ms (σ=5.1 ms) | 84.0 ms (σ=1.1 ms) |
| On-chain view call | 15.2 ms (σ=8.8 ms) | 60.4 ms (σ=22.8 ms) | 176.7 ms (σ=34.3 ms) |
| Cross-validation | Match | Match | Match |
| Total gas | N/A | 27,478,854 | 27,478,959 |

Off-chain verification cost is practically the same on all three environments. On-chain view calls become more expensive on the public testnets because of network round-trips, but the total gas usage is almost identical between Sepolia L1 and Arbitrum Sepolia L2. The cross-validation column confirms that the Solidity contract and the off-chain JavaScript implementation produce the same Merkle root for every dataset.

### 4.4.4. Scaling analysis

The scaling exponent is defined as $\alpha = \log(T_2/T_1) / \log(N_2/N_1)$, where T is mean verification time and N is the dataset size. $\alpha = 1.0$ corresponds to linear scaling. Both blockchain Merkle construction and

the SHA-256 baseline scale linearly ($\alpha \approx 1.0$). This matches the O(n) symmetric-hashing complexity which both paths share.

Table 5
Scaling analysis: empirical complexity exponent $\alpha$ for blockchain Merkle construction.

| Transition | System | Time Ratio | Data Ratio | $\alpha$ |
|---|---|---|---|---|
| 1K → 10K | Blockchain | 11.11× | 10× | 1.05 |
| 10K → 100K | Blockchain | 9.77× | 10× | 0.99 |

#### 4.4.5.

#### 4.4.6. Single-record proof verification

The full-dataset results above cover O(n) verification. The main advantage of a Merkle tree over a hash chain is single-record verification via inclusion proofs, which costs O(log n). For this measurement, Merkle proofs were generated for a representative leaf (the middle index) at each dataset size.

Table 6
O(log n) proof verification (100 runs, first 2 discarded as warmup).

| Dataset | Proof Depth | Off-Chain (Mean) | On-Chain Gas | Avg Gas/Level (Total ÷ Depth) |
|---|---|---|---|---|
| 1,000 | 10 | 410 µs (σ=95 µs) | 37,220 | ~3,722 |
| 10,000 | 14 | 296 µs (σ=45 µs) | 42,340 | ~3,024 |
| 100,000 | 17 | 394 µs (σ=75 µs) | 46,060 | ~2,709 |

Off-chain proof verification stays under 0.5 ms at every dataset size, which confirms sub-millisecond O(log n) behavior. The non-monotonic timings come from V8 JIT warm-up and microbenchmark noise. On-chain gas grows by about 1,280 per additional tree level. The average gas per level falls because the fixed per-call overhead is spread across more levels. The proposed approach is practical for daily operations on Layer 2 networks such as Arbitrum [19].

### 4.5. Discussion

The model choice depends on what the deployment actually needs to defend against. In single-organisation deployments with an internal audit and a trusted database operator, Model A is cheaper and avoids gas costs. So if all participants of the system trust their internal operators, there is no need to switch to a more complex and expensive solution.

If non-repudiation of individual records is more important than completeness, Model B becomes more useful. An example here is regulatory attestations, where liability is bound to each signer's record. In such systems silent deletions are usually a smaller threat than the lack of accountability for what is signed.

Model C should be used in cases when no participant can be trusted to administer the store, and the system also has to detect deletions. This is exactly the situation where Model A and Model B do not work.

The costs of Model C can be further reduced by using cheaper public chains or private blockchains, where the trade-off shifts in favour of Model C. The verification of these results on other blockchains is still required and could be the aim of future research.

The measured numbers also show that the overhead of blockchain integrity is smaller than it is often assumed. Per-record ECDSA verification, which is the standard non-repudiation approach in regulated industries, is slower than the proposed Merkle construction in every measured configuration. This is also confirmed by pharmaceutical traceability pilots [20], where blockchain-based verification was workable for last-mile compliance under DSCSA at low per-unit cost.

Limitations. The experiments used synthetic datasets. Also, the experiment covered only Ethereum-compatible networks (Hardhat, Sepolia, Arbitrum Sepolia). Other networks based on blockchain may be much cheaper and more suitable for the proposed system (Solana, TON and etc).

## 5. Conclusions

In this paper a formal definition of integrity completeness was introduced for shared supply chain data, and three trust models were analyzed under an adversary which controls the data store. It was shown (Proposition 1) that only the third model, which is blockchain with application-level Merkle trees, achieves integrity completeness. The first two models cannot detect deletions and do not work under this threat model. The four-contract EVM architecture proposed in this paper realizes the third model in practice. Access control, business logic, integrity verification and audit logging share a single atomic transaction, so a valid state can be reached only together with a matching integrity proof.

There are also several practical results which follow from this work. In practice the architecture changes two things. It increases what can be detected: silent record deletions, which Models A and B miss, are now caught by a root mismatch. It also lowers the cost of integrity verification. The deletion-detection guarantee can be achieved on existing public blockchains without paying the per-record ECDSA cost which traditional non-repudiation systems require, and on-chain Merkle construction is faster than per-record verification regardless of which secp256k1 library Model B uses. For single-record checks the cost drops from O(n) to O(log n), so a proof verifies in sub-millisecond off-chain time and 37,000-46,000 gas on-chain.

Future researches should compare the proposed approach with non-EVM blockchains (such as TON, Solana, NEAR) and other consensus types. Also, smart-contract upgradeability and the use of zero-knowledge proofs for confidential supply chain data could be studied in the future works.

The Solidity contracts, JavaScript off-chain implementation, benchmark scripts, synthetic dataset generator, and raw measurement logs will be available after publication.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

## Declaration on Generative AI

During the preparation of this work, the authors used GLM-5 in order to: Grammar and spelling check. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

## References


[1] Wang, Y., Han, J.H., Beynon-Davies, P.: Understanding blockchain technology for future supply chains: a systematic literature review and research agenda. Supply Chain Manag. 24(1), 62–84 (2019)

[2] Khan, H.U., Sohail, M., Nazir, S., Hussain, T., Shah, B., Ali, F.: Systematic mapping study of blockchain integrated supply chain management. Secur. Commun. Netw. 2024, Article ID 8884339 (2024). https://doi.org/10.1155/2024/8884339

[3] Sri Vigna Hema, V.: Blockchain implementation for food safety in supply chain: a review. Compr. Rev. Food Sci. Food Saf. 23(5) (2024). https://doi.org/10.1111/1541-4337.70002

[4] Boyson, S.A.: Cyber supply chain risk management: revolutionizing the strategic control of critical IT systems. Technovation 34(7), 342–353 (2014)

[5] Buterin, V.: Ethereum: a next-generation smart contract and decentralized application platform. White paper (2014)

[6] Hyperledger Foundation: Hyperledger Fabric documentation (2024)

[7] OpenZeppelin: Contracts: MerkleProof library. GitHub repository, release v5.0.0 (2023)

[8] OpenZeppelin: Contracts: on-chain MerkleTree library (utils/structs/MerkleTree.sol). GitHub repository, release v5.1.0 (2024)
[9] Wood, G.: Ethereum yellow paper: a secure decentralised generalised transaction ledger. Shanghai version, commit efc5f9a (2025)
[10] Karaduman, Ö., Gülhas, G.: Blockchain-enabled supply chain management: a review of security, traceability, and data integrity amid the evolving systemic demand. Appl. Sci. 15(9), 5168 (2025). https://doi.org/10.3390/app15095168
[11] Wu, J., Xie, L., Li, X.: Security vulnerabilities in Ethereum smart contracts: a systematic analysis. arXiv:2504.05968 (2025)
[12] Sytnyk, R.S.: Models and methods for organization and ensuring data integrity in information system registries. Ukrainian State University of Science and Technology, Dnipro (2025). (in Ukrainian)
[13] Sytnyk, R., Hnatushenko, Vik.: Data flow management in information systems using blockchain technology. Naukovyi Visnyk Natsionalnoho Hirnychoho Universytetu 3, 142–148 (2024). https://doi.org/10.33271/nvngu/2024-3/142
[14] Liu, Z., Ren, L., Feng, Y., Wang, S., Wei, J.: Data integrity audit scheme based on quad Merkle tree and blockchain. IEEE Access 11, 59263–59273 (2023)
[15] Papamanthou, C., Srinivasan, S., Gailly, N., Hishon-Rezaizadeh, I., Salumets, A., Golemac, S.: Reckle trees: updatable Merkle batch proofs with applications. Cryptology ePrint Archive, Paper 2024/493 (2024). https://eprint.iacr.org/2024/493
[16] Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on Ethereum smart contracts (SoK). In: POST 2017. LNCS, vol. 10204, pp. 164–186. Springer, Heidelberg (2017)
[17] Feist, J., Grieco, G., Groce, A.: Slither: a static analysis framework for smart contracts. In: Proc. 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), pp. 8–15. IEEE (2019)
[18] Benet, J.: IPFS: content addressed, versioned, P2P file system. arXiv:1407.3561 (2014)
[19] Bousfield, L., Bousfield, R., Buckland, C., Burgess, B., Colvin, J., Felten, E.W., et al.: Arbitrum Nitro: a second-generation optimistic rollup. Offchain Labs Whitepaper (2022)
[20] Chien, W., de Jesus, J.P., Taylor, B., Dods, V., Alekseyev, L.Yu., Shoda, D., et al.: The last mile: DSCSA solution through blockchain technology: drug tracking, tracing, and verification at the last mile of the pharmaceutical supply chain with BRUINchain. Blockchain Healthc. Today (2020). https://doi.org/10.30953/bhty.v3.134